THE USE OF AN AUXILIARY COMPUTER WITH A GRAPHIC DISPLAY AS AN ON-LINE DEBUGGING AID

Elton Truxton Ashby, Jr.

United States
Naval Postgraduate School

THESIS

THE USE OF AN AUXILIARY COMPUTER WITH A GRAPHIC DISPLAY AS AN ON-LINE DEBUGGING AID

by

Elton Truxton Ashby, Jr.

Thesis Advisor: George E. Heidorn

June 1971

Approved for public release; distribution unlimited.

The Use of an Auxiliary Computer with a Graphic Display as an On-line Debugging Aid

by

Elton Truxton Ashby, Jr.
Lieutenant, United States Navy
B.A., University of Okalhoma, 1965

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1971

ABSTRACT

Often, the most time consuming and costly evolution in the development of computer programs and systems is the testing of the programmer's logic. There are many tools and techniques available which aid the programmer in detecting logic errors, but all have characteristics which limit their usefulness. The objective of this research was to develop a system which uses an Adage AGT-10 as an auxiliary computer with a graphic display to provide facilities for monitoring a program which is running on the XDS-9300 computer. The system developed and implemented enables the programmer to stop the execution of his program, display the memory contents of the 9300, change the memory contents as required, and then continue the execution of his program. The thesis includes information on the use of the system and a detailed discussion of the implementation.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

In the development of computer operating systems and programs a large portion of system and programmer resources is devoted to the detection and correction of errors within the routines under development. These errors in programming may be put into two categories, syntax errors and logic errors.[1]

Errors of the first category result from a misunderstanding of the rules of the language in which the program is written. These mistakes in syntax, by definition, result in an error in compilation or in assembly of the program and are, therefore, detectable prior to execution of the generated code. The presence of this type of error is usually made obvious to the programmer by the assembler or compiler.

The second and more costly type of error results from faulty programming logic. The errors in logic are not detectable until during program execution and are sometimes not detected until the program has been executed many times. There are many tools and techniques available to aid the programmer in his attempts to locate and correct these errors, but there are weaknesses inherent in each of the available methods. In order to alleviate the problems associated with these methods presently in use, new approaches are needed to aid the programmer in testing and correcting his program logic.

---

[1] A third category consisting of various errors resulting from mistakes in the transcription of program files will not be considered because these errors do not constitute programming errors as such. Normally they are no more than irritating, and little expenditure of programmer time is required to detect and correct them.

A. OBJECTIVES

The objective of the research reported on in this thesis was to develop a system which uses an Adage AGT-10 as an auxiliary computer with a graphic display to provide facilities for monitoring a program which is running on the XDS-9300 computer. A secondary objective was to provide easily readable documentation on the programming of the ADAGE AGT-10 graphics terminal and to develop control and interface routines which could be used in other applications. The system was to be simple to operate, provide information in an easily understood form, and interfere as little as possible with the execution of the user's program. The system which was designed and implemented allows the programmer to stop the execution of his program, display the memory contents of the 9300, change the memory contents as required, and then continue the execution of his program.

B. ORDER OF REPORTING

This paper begins with a general discussion of the methods used by a programmer to test and correct his programming logic. This is followed by a brief discussion of the equipment for which the system under discussion was designed. Section IV consists of a user's guide for the operation of the system. It is ordered so that it would provide a useful reference for anyone who would wish to apply the system to his programming problems.

Section V provides a detailed explanation of the techniques used to control and interface the two computers and to provide the graphics display. The next section discusses the application of these techniques as they are used in the system. The paper is concluded with the presentation of some ideas on the application, expansion and further development of the system.

II. DEBUGGING TECHNIQUES PRESENTLY IN USE

Before looking at the system which has been developed, it will be helpful to consider the techniques and tools presently used by programmers to detect and correct his errors and to discuss the attributes of these systems which limit their usefulness in many commonly occurring situations.

The most common error detection method used by a programmer is to simply execute the program to completion. The programmer will normally gather a representative set of test data for which the expected result is known and use this data to test his program logic. This data set will be specifically designed to try special and unusual conditions which could arise when the program is used for production purposes. It will usually contain the largest and smallest numbers expected in the production stream; and some erroneous data may be introduced into this set in order to check the workings of internal test and recovery routines. He will take the output obtained from the run, compare it with expected results, then trace backward through his routines in an attempt to locate the errors in logic, if any occurred. He will repeat this procedure until the expected results are obtained from the test data. This method is commonly used by novice programmers, by students, and by almost all programmers in the development of small programs. It is particularly wasteful of programmer's time and of equipment resources. Normally, a complete run is required to find a single error, and there is always the possibility that the programmer will introduce new errors while attempting to correct those which have been detected.

The programmer may increase the efficiency of this method somewhat by including statements within his program which print interim results. He may, for instance, print the values of indices and loop counters each time a loop is executed, and the values of local variables upon each exit from a routine. He may print a message to himself at various points within the code so that he can retrace the various branches and procedure calls, and the sequence in which his routines were entered. This may speed up the process by lessening the number of runs required, but the programmer is left with the task of tracing backward through a mass of information in order to find his errors.

Another, slightly more sophisticated, variation is to test one routine or block of logic at a time. The programmer may accomplish this by inserting break points within the code. The programmer will execute his program down to a break point, print out results and stop execution. He will then correct the logic errors as he finds them and retest the block of code. This is repeated until he has a particular section of his code working correctly. He will then remove the break point and insert another at a later location and repeat the process. After all of the code sections are corrected, he will make the necessary changes to interface these sections and test the interfaces again, by using a test set discussed above.

A tool which may be employed in conjunction with this general technique is the memory dump. A dump consists of a printing of the current contents of large blocks of internal memory. By using a memory dump, the programmer is able to compare the state of the computer at the break point with the state desired. He may use this tool to aid him in finding such errors as those resulting in negative loop counters or overwritten

code. This may be the only method available which will allow him to locate such insidious "bugs" as overwritten interrupt pivots or system routines. Because a memory dump generally contains more data than is required and it is difficult to analyze, it is again wasteful of both time and resources and the area of memory which is dumped may not contain the errors.

On-line systems have been developed which allow the programmer to apply the above mentioned techniques in a more convenient manner. These systems provide means whereby the programmer, during the development phase of his programming, may specify that certain parameters be displayed; dumps be printed at certain times; and that entry point names and variable names be cross-referenced and indexed. The most elaborate and sophisticated of these on-line aids are trace routines. Trace "packages" are available which allow a programmer to follow the logic as the program is run and to record the various steps executed and the various branches taken by the running program, but these routines have serious limitations. In particular, the resources of the system used are required in order to run the trace and diagnostic routines. There is an inevitable overlaying of memory when these routines are loaded, and this may involve a large portion of available memory. In addition, there is a necessity for the allocation of input/output devices to these routines. Therefore, the programmer is restricted in the types of routines upon which he may apply these techniques. If his program were to use all of the resources of the system upon which his program is run then it would be impossible for him to apply the diagnostic systems which are available to him. In other words, these diagnostic packages are unusable precisely where they are most needed. They may not be used when extremely large programs using the majority of peripheral devices are being developed.

The critical resource in most cases is memory. The utility, dump, and output routines are sometimes quite large and complex. The programmer must either overwrite large portions of memory in order to load these routines, or he must dedicate a large portion of memory resources for their use. This may be unacceptable if the work itself is systems work which is concerned with loaders and resource allocation mechanisms. The changes in memory content resulting from the execution of an on-line package may increase, rather than diminish the complexity of the task.

In an academic environment these techniques are of limited value primarily because they require the student to have special knowledge beyond what he has at the time he is writing the program. The beginning student should not be expected to concern himself with complex methods for obtaining the information needed to solve his problem. The use of diagnostic packages and core dumps requires some level of competance above that expected of a user who is learning to write a program and to test it. Ideally, a system used by a student would allow him to experiment, test, change his methods, and follow where his experimentation leads him. With the present methods there is an inevitable lag between the execution of his experiments and his obtaining the results.

It may be seen, then, that the systems presently in use have disadvantages associated with them which greatly restrict their value for both the novice and professional programmers. The system discussed in this paper represents an attempt to alleviate these problems by providing the programmer with the assistance of a graphics oriented computer to aid him in the application of his test and correction techniques. Because the system resides in a separate computer, the programmer is relieved of the concerns associated with the temporary introduction of system routines

into the memory, and the graphics capability makes the system truly on-line by providing immediate response to the user's queries.

III. FACILITIES AVAILABLE

The system being reported on was designed to run on the XDS-9300, AGT-10 interactive graphics system at the Naval Postgraduate School. The programs normally run on this system are student-written FORTRAN jobs which make use of the interactive and graphical capabilities of the two machines. The students operate the computers in a laboratory environment "hands on". The laboratory contains the XDS-9300 with associated peripheral equipment and two of the AGT-10 graphics terminals. Figure 1 depicts the equipment available in the laboratory and the conceptual relationships among the computers.

A. HARDWARE

The XDS-9300 machine is a medium-speed second generation computer with 32,768 core locations of 24-bit words. It is provided with a drum, two tape drives, page printer, card reader, paper tape punch, and paper tape reader.

The computer used to monitor the XDS-9300's operation is the ADAGE graphics terminal. The ADAGE machine is somewhat unusual in a number of respects. It should not be considered to be merely an "intelligent terminal"; it is a quite sophisticated third generation computer system, limited only by internal capacity. In fact, the machine has an extensive software support system, and a large library of routines which provide convenient file manipulation and editing. Probably the most unusual feature of this machine is the amount of control available to the assembly language programmer over the input/output devices and channels.

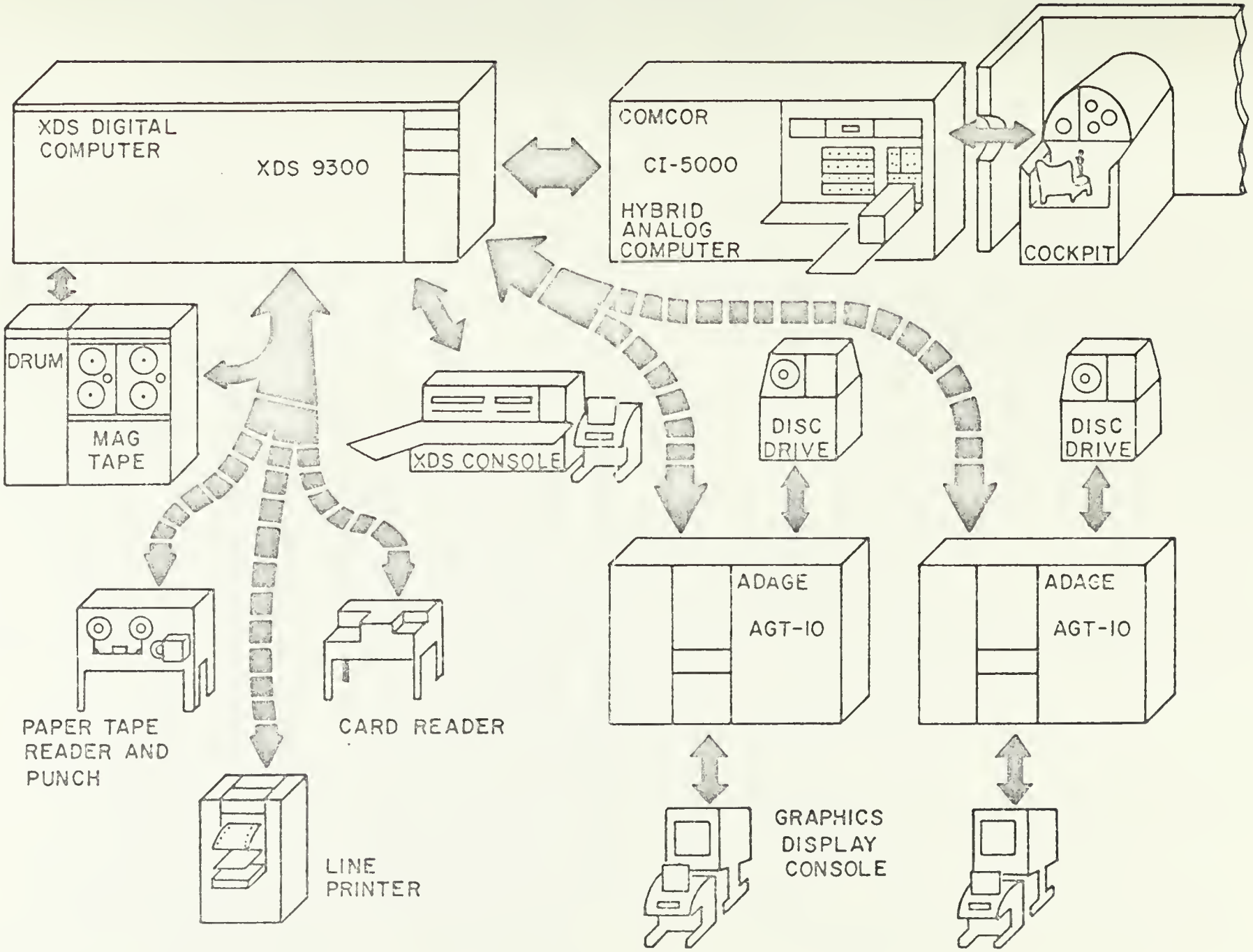


Figure 1 THE EQUIPMENT AVAILABLE

The input/output devices associated with this machine consist of a cathode-ray-tube graphics display system with both a vector generator and character generator, a teletype terminal for character input/output, and a set of push-button switches for interactive and control functions.

The XDS-9300 associated memory interface channel is used to provide communication between the two machines. The channel is directly addressable from the graphics terminal, and is easily controlled by a program residing in the ADAGE machine. One distinct advantage with this system configuration is that the interface allows the ADAGE graphics terminal to have priority over the XDS-9300 in accessing the 9300's memory. This attribute of the system reduced the complexity involved in the synchornization of the two computers.

B. SOFTWARE

Most of the programs and projects run on the system make use of library routines which allow the students to display various line drawings and text listings on the screen of the graphics terminal. These routines may be called from a FORTRAN program in execution on the XDS system, and, when coupled with a system routine which is executed in the terminal computer, allow quite complex graphical displays.

Since most of the student projects are interactive as well as graphical, it is also necessary that facilities be available to allow the accessing of values by variable name at run time. To make a variable name available, the user may specify a NAME LIST attribute within his FORTRAIN program. He may specify that certain variables or that all variables be appended to the resident NAME LIST.

IV. FROM THE USER'S VIEWPOINT

The programmer who desires to apply the system under discussion need have little more information and experience than that required to compile and execute his program on the XDS-9300. In fact, more detailed knowledge of the XDS-9300 software is required to obtain trace listings and printed dumps than is required to operate this system. The user does not, for instance, need to know the entry point address or the NAME LIST start address for his program, and he need have little familiarity with absolute machine code because memory contents are displayed in mnemonic format as well as in octal format.

If his program is one which makes use of the interactive graphical routines available to him (which is usually the case), he may still use this system. He may display his graphical outputs on one AGT-10 and use the other to aid him in testing his logic. The system in no way reduces the interface capabilities of the XDS machine; nor does it limit the types of program which may be run.

In order to apply this system to his particular program, the user must include a four card patch deck ahead of the routines to be compiled or assembled. The patch results in a change in the contents of one word in the XDS resident software.[2] Figure 2 represents a typical student job deck, including the cards required to effect the patch. The "ΔAGT" card must be included after the patch deck. After the user has added the required cards, he may then compile, load and execute in the same manner as he would if he were not using the system.

---

[2]Note that sense switch number two on the XDS-9300 console must be on at load time if a patch is to be used.

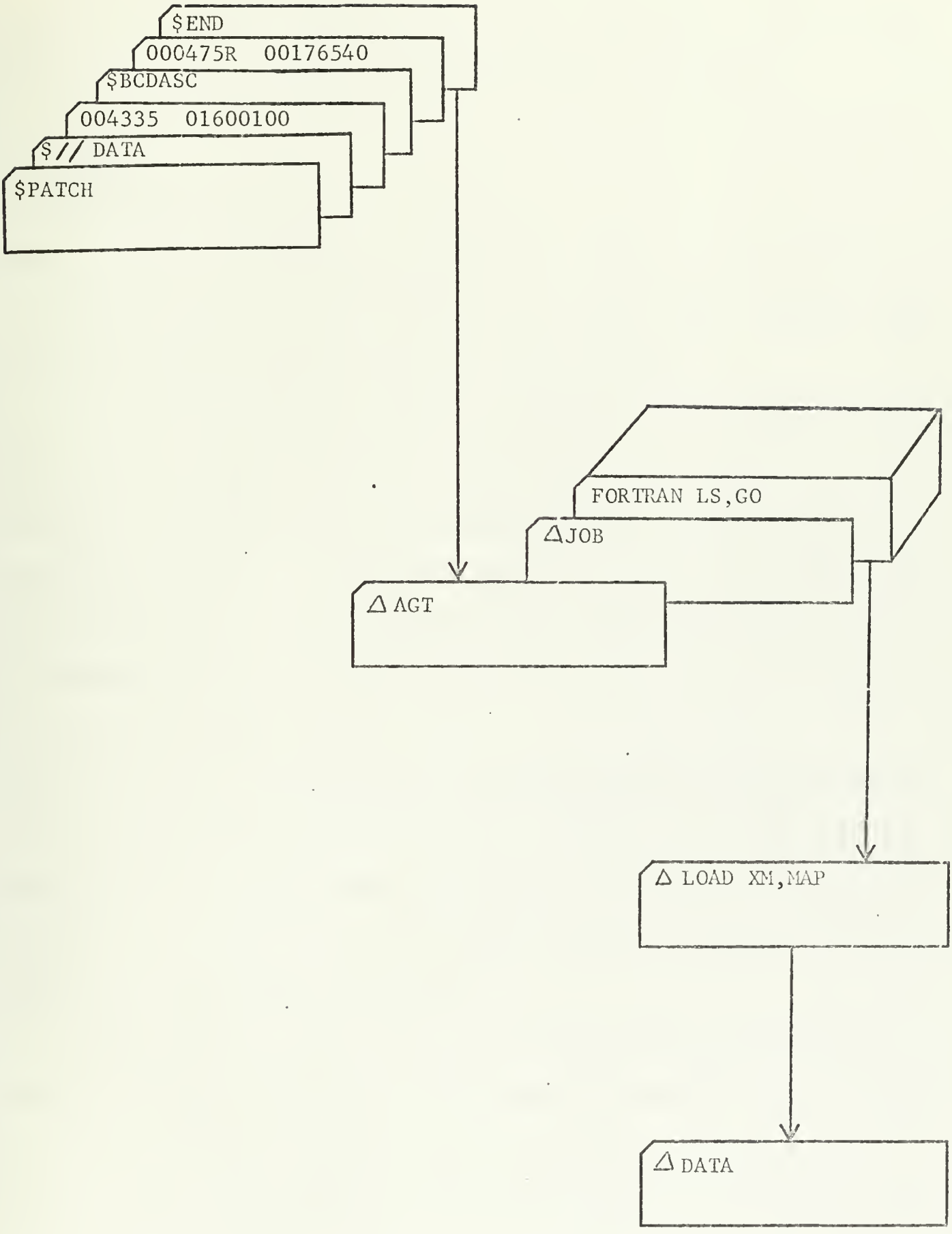


Figure 2 STUDENT JOB DECK

On the AGT, the user must first type the line START("DISP",PVV)!, where PVV is the pack number and two digit volume number upon which program DISP resides. The AGT will respond with the following:

```
REQUIRED:
OFST
ICHTY
NEW PACK, VOL=
```

This is a request that the user provide the loader with the location of two systems routines which are called from program DISP. The user must respond with the information, again in the form PVV.

When the user desires to stop the execution of his program and apply the routines, he will type DISP!. When he does this, execution of his program on the XDS machine will be halted and the first 40 words of his program will appear on the screen. The first word of this display will be the entry point into his executable code. Each word of the display is presented in the following format:

```
LLLLL: NEM AAAAA 00000000
```

The L field is the core location of the word expressed in octal digits; NEM is the META-SYMBOL equivalent for the operation code portion of the word; the A field is the address field of the word; and the 0 field is the contents of the word in octal format. Figure 3 illustrates the appearance of the screen at this time.

At this point, the user may, by pressing various combinations of function switches, apply many of the techniques currently in use to test the logic of programs. He may display his variable names and their decimal values; move forward or backward in core, displaying memory contents in 40 word blocks; he may specify a start address and display a sequence of blocks starting at that point; he may change the contents of a word in the XDS-9300 memory. When he has completed his work, he may

continue execution of his program on the XDS machine. The process may be repeated as many times as desired.

In order to explain how the user may apply the different functions available, a detailed discussion of the response of the system to each function switch will now be presented.

A. FUNCTION SWITCH ONE--SERVICE REQUEST

The user considers function switch number one to be a service request switch. The routine associated with this switch allows the programmer to choose between two modes of operation when he selects some of the other functions. He may choose a slow-continuous mode or a discrete mode in many of the functions available to him. When function switch one is depressed simultaneously with another function switch, that function will be repeated until the switches are released. If he chooses to operate in the discrete mode, he will first request a service by selecting switch one then press the switch associated with the desired function.

The incorporation of the two modes of operation has proven to be very useful where, for instance, large portions of the memory must be scanned. The user may select the slow-continuous mode for displaying successive blocks of memory. He would do this by selecting both function switch five and function switch one. This will result in a sequence of successive blocks being flashed on screen, approximately one every second. As he approaches the area of interest in memory, he will de-select function switch one and operate in the discrete mode by alternately depressing function switch one and function switch five. This will result in the display of the next block each time function switch five is depressed.

```
21237:  BRM 02345   00302345
21240:  LDP 31445   02631445
21241:  STD 41111   074411111
21242:  BRM 23222   00323222
21243:  LDA 24444   01624444
21244:  ADD 24446   00624446
21245:  STA 24444   07524444
21246:  BRM 23222   00323222
21247:  NOP 00000   01000000
21250:  LDP 33333   02633333



21275:  HLT 00000   00000000
21276:  HLT 00000   00000000
21277:  HLT 00000   00000000
21300:  HLT 00000   00000000
21301:  HLT 00000   00000000
21302:  HLT 00000   00000000
21303:  HLT 00000   00000000
21304:  HLT 00000   00000000
21305:  HLT 00000   00000000
21306:  HLT 00000   00000000
```

Figure 3 SCREEN DISPLAY

B. FUNCTION SWITCH TWO--VARIABLE NAMES

As noted earlier, the programmer may specify that certain variables or that all variables within each routine be placed in the NAME LIST. When function switch number two is selected by the user, the variable names within the NAME LIST and their decimal values are displayed on the screen. Figure 4 illustrates the appearance of the screen when this function switch is selected.

The first section of this display contains the NAME LIST associated with the main routine. Each subsequent section is headed by a subroutine name, followed by the NAME LIST for that routine. The slow-continuous mode of operation for this function is not appropriate and has not been implemented.

C. FUNCTION SWITCH FOUR--CONTINUE EXECUTION

After the programmer has completed his work for the iteration and has made the changes desired within the XDS-9300 memory, he will then select function switch number four. This will terminate the system resident in the ADAGE computer and cause the XDS-9300 to continue in execution from the point where it was executing prior to the interruption from the ADAGE.

If the user desires to halt execution and take another look inside the XDS memory, he must reinitialize the programs on the ADAGE and, by typing the appropriate commands, he may again use the system.

D. FUNCTION SWITCH FIVE--NEXT BLOCK

When the programmer selects function switch five, the current screen display will be replaced by the next 40 word block of contents from the XDS-9300 memory. This switch is used when the programmer wishes to scan

```
XXX       00003E00004
I         00049
YYY       00099E00009
JJ        00082
K         00001

LOOP      00000
X         00001E00001
I         00099
Z         00004E00003

SUB       00000
ABC       01111E01111
I         00001E00001
```

Figure 4 NAME LIST DISPLAY

sequentially through a section of memory. Both the slow-continuous mode and the discrete mode are available. Figure 3 shows a sample of the screen display for this case.

E. FUNCTION SWITCH SIX--LAST BLOCK

The routines associated with this switch allow the user to move backward through memory, as switch five allows him to move forward. Both modes of operation are available and may be used in conjunction with this switch.

F. FUNCTION SWITCH SEVEN--DISPLAY ADDRESS

This function allows the user to choose a starting address for displaying the memory content blocks. When this switch is selected, the teletype responds with the queue line START ADDRESS=. The user then types a five digit octal number. Immediately after typing the fifth digit, the block starting with the typed address will appear on the screen.

G. FUNCTION SWITCH NINE--CHANGE CONTENTS

This function allows the user to change memory contents within the XDS-9300. When the switch is depressed, the line ADDRESS= will be typed on the teletype. The user will respond with a five digit address. After the last digit is typed the line NEW CONTENTS= will appear. The user then types an eight digit octal number. This number is stored in the specified location in the memory. In order to use this function to change the value of a variable, the user would first have to locate the variable name by displaying the memory block containing the NAME LIST. He would note the address in the word immediately following the variable name and enter the new contents starting at that address.

V. IMPLEMENTATION

In order to describe the implementation of the functions made available by the system, the software which has been developed to utilize the various attributes of the ADAGE machine and of the interface will be discussed in this section. A complete understanding of the various graphical control, and interface routines will be necessary in order for the reader to follow the programming of the system. The logic employed within the service routines is straightforward, but the coding within the routines which interface the two computers and which provide the graphical and control functions is system dependent and is not easily followed.

A. CONTROL

The first to be discussed will be the scheme used to control execution of the XDS-9300 computer. This is accomplished by the use of the priority interrupt system and a small stored program within the 9300 memory.

1. Interrupt System

In the XDS machine, the two memory locations 43_8 and 44_8 are associated with AGT number 1 and number 2 respectively. When the command OPIO 43001 is executed in the AGT, an interrupt within the XDS machine is generated. This interrupt causes the contents of the location associated with the ADAGE (either 43 or 44) to be loaded into the instruction register and executed. When the AGT parameter is specified within the user job deck, which is the case when the DISP routines are used, these locations contain a Mark Place and Branch instruction into a resident routine. When this instruction is executed, the current contents of the

location counter is stored into the location specified by the address field of the instruction and the address field plus one is loaded into the location counter. In the case of this program, when an interrupt from the monitor ADAGE is sensed, the user's program is interrupted, the place in execution at which the interrupt occurred is stored in a location in the resident routine and a branch to the routine is taken. The resident system routine provides the interface for the FORTRAN callable library routines mentioned earlier. This routine stores the A, B, and index register contents, then accesses a three word flag buffer located in high core within the XDS machine. The buffer for AGT1 begins at 77764_8, and that for AGT 2 at 77767_8. These two buffers are referred to as Status Words In (SWI) in the documentation and serve as a mailbox for the system communication routines.

After obtaining the contents of the appropriate buffer, the system routine performs a series of comparisons to determine the response required by the ADAGE. If the status word is found to be invalid, then, under normal circumstances, a halt instruction is executed. It is this halt instruction which is changed by the patch deck. When the resident software is loaded, this instruction is changed to a BRU 76540 (Branch to 76540 unconditional). Location 76540 is the beginning of a tape-to-disk transfer buffer which is not available for use by FORTRAN programs. With this patch in place, it is only necessary to insure that the SWI is invalid in order to gain control of the XDS computer. When the user begins execution of the system by typing DISP! at the ADAGE console, an invalid flag is loaded into the SWI, an eight word program containing a trap at the last location is loaded into the XDS memory beginning at 76540, and an interrupt is sent to the XDS machine. The user may then

apply the routines described above, to observe and change the contents of the XDS memory. When he desires to continue execution of his program in the XDS, he calls a routine which overlays the trap location with an unconditional branch back into the resident routine. This branch is to the beginning of a block of code which restores the registers, clears the original interrupt, and executes an indirect branch through the routine entry point. This results in a return to the user's program at the point where execution was interrupted.

B. INTERFACE

The routine written to effect memory transfers for the system under discussion is perhaps the most complex and difficult to follow of all of the routines in the program. For this reason, a detailed flow chart has been included as Figure 5. It will aid the reader to follow the logic on the flow chart as he reads the explanation of the routine.

It should be noted that only the ADAGE is active in all data transfers. The XDS machine has no facility for initiating transfers in either direction. It is also important to realize that while transfers are on a word-for-word basis, the word lengths for the two machines are different. The ADAGE has a 30-bit word while the XDS has a 24-bit word. This results in a truncation of the high order six bits when a transfer is made into the XDS-9300. When the transfer is into the ADAGE memory, the high order six bits are filled with zeros and the low order 24 bits contain the transferred words.

1. Interface Pivots

To effect a memory transfer, it is necessary to make four items of information available to the interface. These four items are the

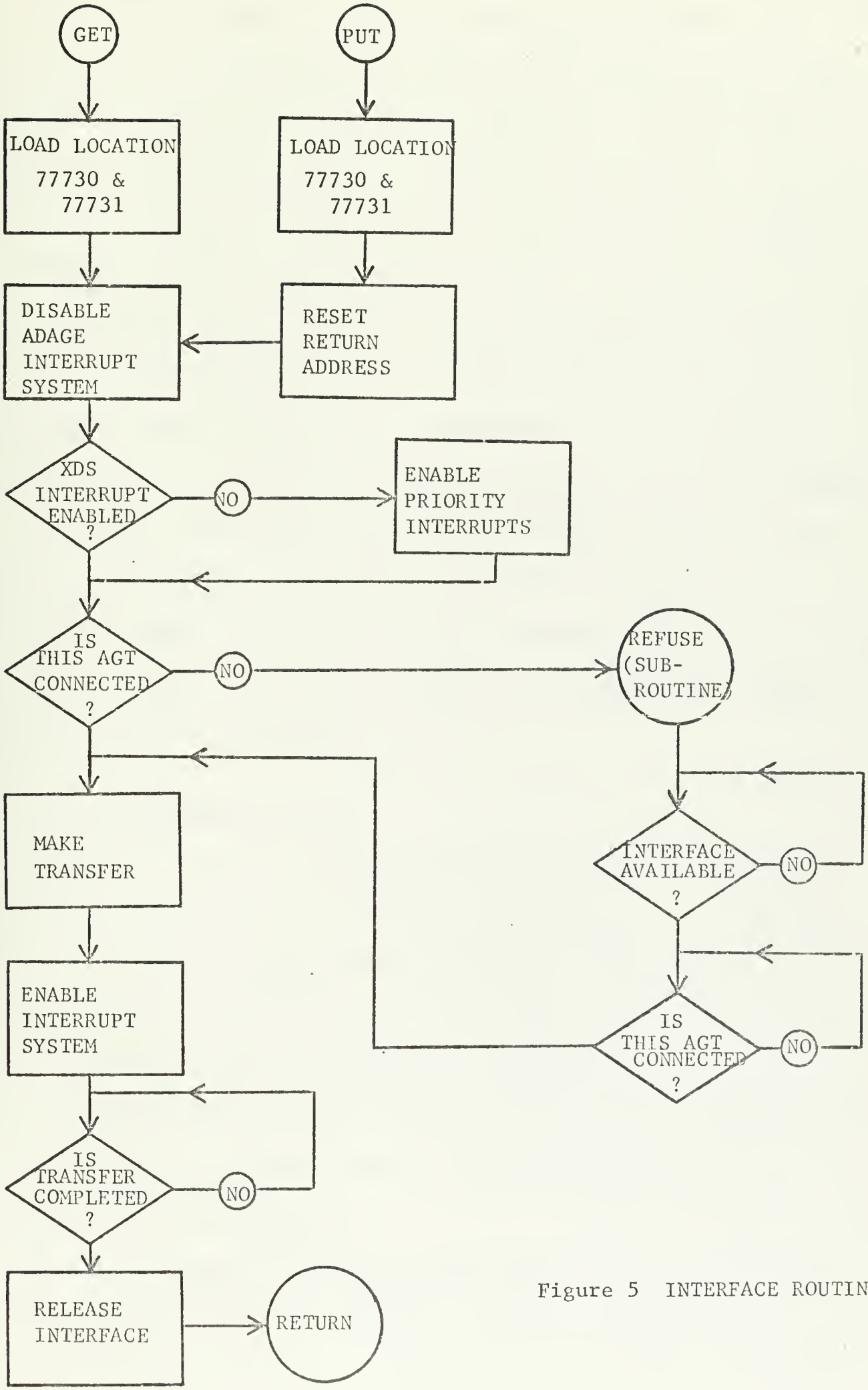


Figure 5 INTERFACE ROUTINE

start location in the ADAGE, the start location in the XDS-9300, the number of words to be transferred, and the direction of transfer. Two ADAGE words are required to contain this information. The first word contains the direction of transfer and the XDS memory address. If bit number seven of this word is set, then the transfer is from the ADAGE to the XDS machine. A zero in this location results in a transfer in the opposite direction. Bits 14 through 29 contain the XDS start address. The format of the second word is similar except that bits zero through 14 contain the number of words to be transferred. Bits 15 through 29 contain the ADAGE start address. Figure 6 illustrates these word formats.

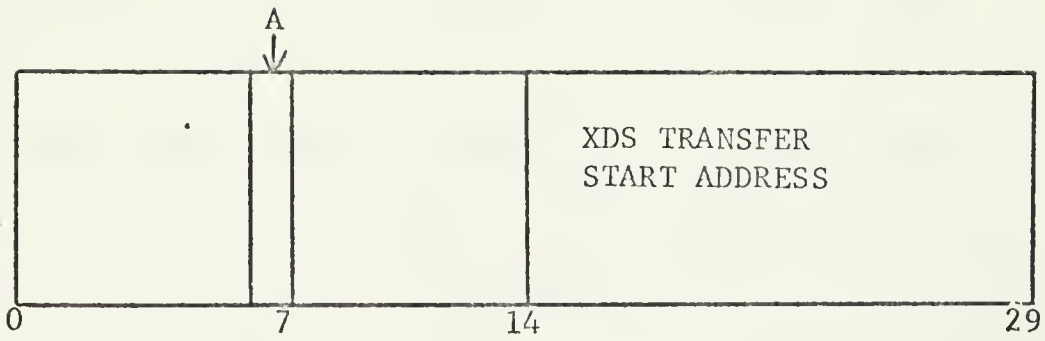
Of course, the interface must have the location of these words before they may be accessed, so under the present implementation, a pointer to word number one must be stored in location 77731_8 and a pointer to word number two stored in 77732_8 in the ADAGE memory before use may be made of the interface.

2. Interface Instruction Set

There is a set of 11 ADAGE executable instructions which allow the programmer to control the interface, and thus, access the XDS memory. In the ADEPT assembly language, these instructions are all of the form OPIO nnnnn, where n is an octal digit. The address field of each command relates the function of the instruction. The first two octal digits identify the device (43 for the XDS-9300) and the last three digits specify the action to be taken. Each of these interface instructions falls into one of two categories.

The first category contains the OPIO instructions which result in an action being taken by the interface. When these commands are

WORD 1



A. Bit 7 indicates direction of transfer
SET - AGT to External
RESET - External to AGT

WORD 2

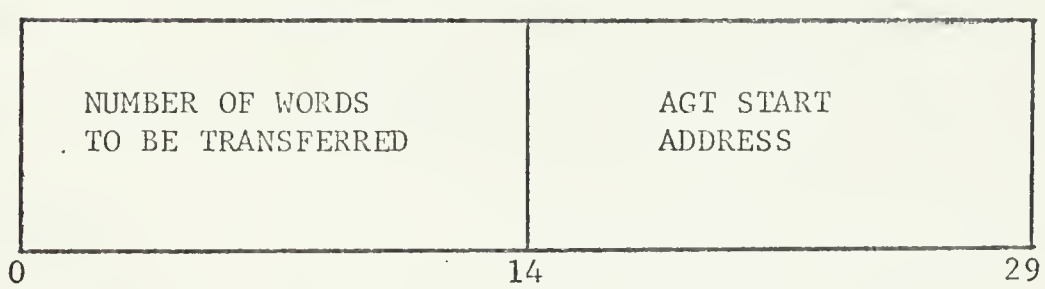


Figure 6 TRANSFER PIVOTS

executed an action such as an interrupt enable, disable, or generation; or a memory transfer takes place.

The commands of the second category result in a test for a specific condition of the interface. These instructions ask of the interface such questions as "Is the interface busy with a transfer?". If the answer to the specific question asked is "yes", then the next instruction in sequence after the test instruction will be skipped. If the test is negative then the next instruction is executed. This arrangement, while not difficult to employ once the idea is grasped, does result in some coding sequences which are rather peculiar in appearance. A sequence which tests continuously until a specific condition is met will appear to the casual reader to be an infinite loop. A good example of this is the two instruction sequence which tests to determine whether the memory transfer has been completed. The sequence consists of the test instruction immediately followed by an unconditional jump to the same test instruction. This sequence will result in a continual testing of the interface until a transfer is completed. When it is completed and the condition is met, then the jump instruction will be skipped and the instruction following it will be executed. Appendix A contains a list of the OPIO instructions and a brief explanation of each.

3. Interface Routine

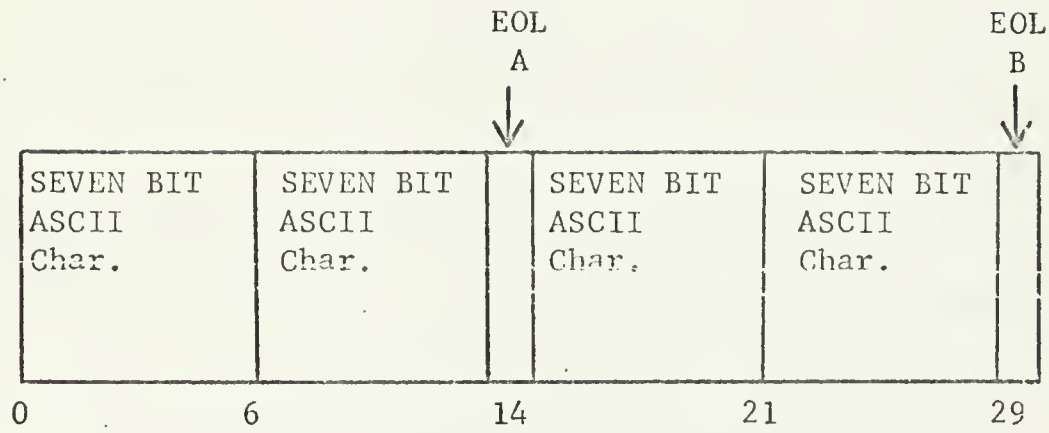
The interface routine, itself, illustrates the fact that several steps are involved in completing a transfer of data from one memory to the other. The routine has two entry points labeled GET and PUT. GET effects a transfer to the ADAGE, while PUT transfers into the 9300. Since the pivot words described earlier contain the direction of transfer, it is not necessary to have unique instructions for transfer in each

direction. Consequently, the code sequences for transfer in either direction are identical, and the same code is executed in order to effect both transfers in the interface routine.

Before the routine is entered via either entry point, it is assumed that the two pivots associated with the entry are set with the desired direction bit, block length, and addresses. In the program, PPVT1 and PPVT2 are associated with the PUT entry and GPVT1 and GPVT2 with the GET entry. When the code is entered via either entry point, the location 77730_8 and 77731_8 are loaded with pointers to the associated pivot words.

If the transfer desired is from the XDS machine to the ADAGE machine, then the routine is entered via GET. The location 77730_8 is loaded with the instruction MD13 GPVT1 and 77731_8 is loaded with the instruction MD13 GPVT2. (It is important to note that the instruction MD13 is not a member of the ADEPT statement set, and must be declared by inserting the statement MD13=33000!H at some point in the program.)

After these locations are loaded, three statements in sequence are skipped. These statement are associated with the PUT entry and will be discussed later. The block of code following this actually effects the transfer. Before it is executed, however, it is necessary to disable the ADAGE interrupt system. The results of the occurrence of an interrupt during a data transfer are unpredictable, but usually, if an interrupt were allowed to occur, then the data being transferred would be garbled. The FPRI instruction disables the interrupt system. After this is accomplished, it is necessary to test to insure that the XDS priority interrupt system is enabled. If the test fails, the next instruction enables the interrupt system, otherwise, the enable instruction



A. Bit Set results in Exit Via 77736_8

B. Bit Set results in Exit Via 77737_8

(Both bits may not be set in same word)

Figure 7 DISPLAY LIST WORD

is skipped. A test is then made to determine whether the user's AGT is presently connected to the channel. If the test fails, a branch to the subroutine RQUSE is executed. This subroutine continually requests usage of the channel until the connection is made, by executing the instruction OPIO 43140 followed by an unconditional jump back to the instruction. The ADAGE documentation does not reflect the fact that this particular instruction is of the test category. A skip is taken, however, if the instruction results in a successful request for usage. The subroutine then tests the connection and exits when this test is successful.

After connection, the instruction OPIO 43020 is executed, this actually results in the data transfer. The routine then tests for completion, enables the priority interrupt system, releases the interface, and exits.

If the transfer is from the ADAGE memory to the XDS memory, then the code is entered through location PUT. The location 77730_8 is loaded with MD13 GPVT1 and 77731_8 is loaded with MD 13 GPVT2. Code is then executed which sets up for the return, and a branch is taken into the transfer code described earlier.

In order to use this transfer routine in programs other than the one for which it was written, a programmer would first have to declare the MD13 instruction and before calling GET or PUT, he would have to load the pivots with the appropriate parameters. He could then call the routine from his program and effect the transfer.

C. CHARACTER GENERATOR

The line character generator, LCG1, associated with each ADAGE is conceptually simple, but several subtleties are involved in its use, which may result in wasted effort by the ADEPT programmer if he is not

aware of them. Basically, when the character generator is turned on it will display a list of ASCII characters once on the screen. It is, therefore, necessary for the program to periodically turn on the character generator and refresh the screen.

1. Display List

The list of characters to be displayed on the screen are stored in a sequential series of ADAGE words in memory. Each word contains four seven-bit ASCII characters in the format illustrated in Figure 7. Bits 14 and 29 are special flag bits which mark the end of the character generator list. When either or these bits is sensed, an end-of-list interrupt will result and an exit from the character generator will be taken. Either, but not both, of these flags may be set within any word of the list.

a. Control Characters

The placement, size, and type of character may be varied by the programmer by the insertion of one of the special control characters into the list. These control characters are described in Appendix B.

If no control character is placed in the list prior to the first character to be displayed, then the first character will be of intermediate size (there are three sizes), intermediate brightness (there are three levels of brightness) and in the center of the screen. In the DISP program the control characters are inserted into the list prior to the first display character to reduce the character size to the smallest, increase the brightness to the maximum and place the first character in the upper left hand corner of the screen. Whether by default, or by use of the control characters, after the first character is placed on the screen, the following characters in the list will appear from left to

right. There is no automatic "line-feed", or "carriage-return" and control characters to start a new line of text must be placed in the list. For the smallest character size, each line consists of 96 characters; there are 40 lines in a full screen display.

2. Application

The character generator must be given two items of information before a list may be displayed. The first, of course, is the location of the display list. This address is placed in absolute location 77735_8. The second item needed is the location of the subroutine which will handle the interrupt generated by the end-of-list flag bits. The interrupt handler for bit 14 is placed in location 77736_8 and that for the bit 29 flag must be loaded into 77737_8. Since exits in two directions are not required in the DISP program, both of these pivots are loaded with the location of a subroutine named RTN.

3. Clock

Once the pivots are set, it is only necessary to periodically turn on both the character generator and the graphics screen in order to display the list. This program makes use of a real time clock, referred to as the frame clock, in order to control the refresh rate. The clock is turned on by setting bit number five in destination number ten within the ADAGE register system. Once the clock is on, an interrupt will be generated every one-sixtieth of a second. The pivot associated with this interrupt is location 77755_8. When the clock interrupt is sensed, a branch is taken to the address contained in location 77755_8. In the DISP program, this location contains the address of a routine called CLOCK. Thus, when a clock interrupt occurs, a routine called CLOCK is executed.

From this point it would seem to be a simple matter to have subroutine CLOCK merely turn on the screen and the character generator, then have the end-of-list interrupt handler turn them both off.

Unfortunately, this approach is not workable for long lists because approximately three-sixtieths of a second are required for the character generator to paint the entire screen. If a clock interrupt were to occur while the character generator is active, the generator would reinitialize and the next character in the list would appear in the center of the screen and would be of size two, regardless of the intended location and size.

Another possible approach, and the one most commonly used, is to count the frame-clock interrupts, and on every third interrupt, start the character generator. When this method is used, one is sure that the worst case, or longest list, may be put on the screen, and that, therefore, lists of all lengths could be displayed. The difficulty is that a three-sixtieths of a second delay between refreshes results in a noticable screen flicker. This flicker is especially annoying and noticable in displays containing dense arrays of small characters.

A method to minimize this screen flicker was incorporated into the program under discussion. As stated above, when the frame clock interrupt is sensed, subroutine CLOCK is entered. The code in this subroutine immediately turns off the frame-clock and turns on the character generator. Then, in the end-of-list interrupt handler, the clock is turned on and the character generator turned off. This method allows the screen to be refreshed at the fastest rate possible, but also guarantees that the character generator will not be interrupted by the clock. A minimum time between refreshes of one-sixtieth of a second is

insured (this is necessary to protect the CRT), but no arbitrary delay between refreshes is established. In practice, this method results in a dense character display with little noticable flicker.

Subroutine CLOCK is easily explained, but subroutine RTN, the end-of-list interrupt handler, is somewhat more complex. When either end-of-list interrupt is sensed, subroutine RTN is executed. The first thing that must be accomplished in this subroutine is to reset location 77730_8 to the head of list location. This is necessary because the character generator actually uses this location as a counter, adding one to the location prior to accessing each word in the display list. One result of this arrangement is that the first word in the list is never displayed. In program DISP, word LOOK + 1 is the first word in the display, while the address of LOOK is the location which is loaded into the pivot. After this pivot is reset, the frame-clock is enabled and the routine is exited.

VI. THE PROGRAM

Now that the input/output, interrupt, and interface routines have been discussed in some detail, it is possible to study the service routines which the system makes available to the user. The first portion of code to be discussed will be the main program. This is followed by a general discussion of the code associated with the programmed function switches. A complete listing of the program is contained in the last section of this paper.

A. MAIN PROGRAM

The program is entered from the monitor at entry-point DISP. The first task performed after the system begins execution is to determine upon which AGT the system is being executed. This is necessary because the eight word program which is executed in the XDS machine periodically is required to interrupt the monitor computer in order to indicate that a task has been completed, and there is a different interrupt generating instruction for each of the two ADAGES. The procedure used to make this determination is to execute the ADEPT instruction OPIO 43104. This is a member of the test category of interface instructions and it asks the question "Is this AGT number 1?". If the answer is in the affirmative, then a skip is executed and the remainder of the program is executed as loaded; if not, then the routine AGT2 is entered. This routine simply changes the code which will later be used in the XDS machine to contain AGT2 interrupts instead of code which will interrupt AGT1.

The next several statements initialize the various interrupt pivots which are used throughout the remainder of the program. After the pivots are initialized, code is placed into the XDS memory which will, when executed, retrieve the entry point of the user's program and the address of the head of the NAME LIST. This code, with an explanation of each command is listed in Figure 8. It was necessary to use the trap program to retrieve these pointers because locations 0 to $20,000_8$ are not directly accessable by the interface. This is due to hardware limitations of the system for which the program was designed. After this code sequence is transferred into the XDS machine via routine GET, an interrupt is generated in the XDS-9300 which results in this code being executed and execution being trapped in a loop within the code sequence. After this code is executed, the pointers which have been loaded and stored are then retrieved from the XDS memory.

Subroutine FN3 is then executed. This subroutine loads a counter with the start address of the user program, and generates XDS executable code which will access the user program in eight-word blocks. This code, listed in Figure 9, is then inserted into the XDS memory, overlaying the code previously contained in the buffer beginning at location 76540_8. It loads eight words of the user's program and stores them beginning at location 76550_8. After the memory locations are moved into the buffer, the ADAGE is interrupted; signalling that the ADAGE may access the code. Since the interrupt generating instruction is contained in an infinite loop, it will result in the ADAGE being continually interrupted. This is prevented by freezing the ADAGE interrupt system immediately upon receipt of the first interrupt from the XDS machine. After this is accomplished, the interrupt code is overlayed with a No Operation instruction. With this never-ending sources of interrupts silenced, it is then

possible to enable the interrupt system and to access the buffer containing the memory contents. The contents are brought into the GBUF buffer, formatted, and loaded into the display list, and the location counter is increased by eight. This is repeated until a full display of 40 lines is obtained. A return to the main program is then executed. In the main program, an infinite loop is entered which continuously samples the function switches and calls appropriate service routines as requested by the user.

B. FUNCTION SWITCH TWO--NAME LIST

At run time, the XDS memory contains a separate NAME LIST for each user routine. The lists are linked by pointers in the first word of each list. A pointer to the head of this chain is accessed by the main program and stored in location CP within the ADAGE memory. When the routine associated with function switch two is entered, an algorithm is executed which accesses the resident NAME LISTS in the XDS-9300 memory. The tabled variable name and the value for each variable is fetched from the XDS memory by directly accessing the XDS memory from the AGT. The variable names are converted, by a table look-up technique, from binary coded decimal to the ASCII character set used within the ADAGE. The current value of the variable is then accessed, converted from octal to decimal, and converted to the ASCII equivalent characters. These, along with the character name and octal location, are placed in the display list. The variable names and values are then displayed on the screen.

C. FUNCTION SWITCH FOUR--CONTINUE

When function switch four is sensed, the code in execution in the XDS machine is overlayed with an unconditional branch instruction which

Figure 8

POINTER RETRIEVAL CODE

	MACHINE CODE	MNEMONIC	PURPOSE
76540:	01000000	NOP	No operation
76541:	01000000	NOP	No operation
76542:	01602124	LDA	Load register A with pointer to user program
76543:	07676551	STA	Store pointer 76551
76544:	41604560	LDA(INDIRECT)	Load A with contents of ADDR in 4560 (NAME LIST)
76545:	07676550	STA	Store first NAME LIST link in 76550
76546:	00176546	BRU	Branch unconditional to this location (trap)

Figure 9

MEMORY BLOCK RETRIEVAL CODE

	MACHINE CODE	MNEMONIC	PURPOSE
76540:	01000000	NOP	No operation
76541:	1176547	LDX(1)	Load index register with -8
76542:	116xxxxx	LDA(INDEXED)	Load register A with memory word
76543:	17676560	STA(INDEXED)	Store word in transfer buffer
76544:	15776542	BRX	Increment index and branch if negative
76545:	00232020	ALC	Interrupt the ADAGE
76546:	00176540	BRU	Branch to top of loop
76547:	00177770	-8	For index register

results in an exit back into the resident system routine. This code is the normal return from the interrupt handler which is resident in the 9300. It clears the interrupt originally generated by the ADAGE, restores the registers, and re-enters the user's program. At this point, the routine resident in the ADAGE may be reinitialized and the user may again look inside his computer.

D. FUNCTION SWITCH FIVE--NEXT BLOCK

The algorithm for placing the next block of code on the screen makes use of the code which retrieves the original block from the 9300. The routine increments the counter used to mark the place in the XDS-9300 presently being displayed and re-executes the code which placed the original block of data on the screen. After the 40 word block has been retrieved, the service routine exits into a loop which continuously checks for a selection of function switch one. If it is selected and function switch five has not been deselected, then the process will be repeated. If function switch one is selected and function switch five is deselected then the main program is entered and all of the function switches will be sampled in turn until another selection is made.

E. FUNCTION SWITCH SIX--LAST BLOCK

The routines providing the facility to view the block preceding the block presently on display on the screen works exactly as does function switch five, except that the counter pointing to the present location in memory is decremented instead of incremented. The slow-continuous mode and discrete mode of operation are provided and implemented as in function switch five.

F. FUNCTION SWITCH SEVEN--DISPLAY ADDRESS

By selecting this function, the user may specify a starting address for displaying the memory blocks. When the function switch is sensed, the program enters the monitor six times in succession. On the first entry into the monitor, the queue line START ADDRESS= is typed on the teletype. This output is accomplished by the use of the system routine OFST. Then system routine ICHTY is entered five times in succession and the program waits for the user to enter a one digit character each time. In the time interval between the striking of the keys, the ASCII input is appropriately masked and shifted. On the fifth character the block location counter is changed to equal the octal equivalent to the input through the teletype. The code which retrieves the memory content blocks is then executed and the new block, starting at the address entered through the keyboard, is displayed on the screen. The appearance of the specified block on the screen is, to the user, instantaneous with his typing the last digit. After the display is placed on the screen the routine is exited.

If the desired block is in high core, it is possible for the user to select a start location which would call for a block containing addresses exceeding core size. For instance, if a user were to specify the octal address 77770, the eighth address in the list would be 100000, which exceeds the memory size. When this happens (as it often does in actual usage) the display "wraps around" memory. In other words, the address displayed following 77777 is 00000. This implementation is true for the functions previously described, and is accomplished by masking all but the least significant five digits from the location counter prior to each memory access.

G. FUNCTION SWITCH NINE--CHANGE CONTENTS

This routine uses the same system routines that are called by function switch seven in order to communicate through the teletype. Once the required address is obtained, it is reformatted into the form used within the XDS memory. A pivot for use by the interface is then created which specifies the address which has been entered. The user then enters the new contents of the XDS word. This word, after it has been reformatted is inserted into the XDS memory by subroutine PUT. Execution then is returned to the main program.

VII. CONCLUSIONS AND RECOMMENDATIONS

The system developed in this research appears to be a useful tool in the area of program development and testing. The ADAGE AGT-10 supplied all of the resources required to execute an adequate debugging package for use with the XDS-9300, and the application of the interactive graphics capabilities of the machine made it possible to produce a system which is simple to operate and which presents needed data in a clear and concise manner. The most valuable characteristic is probably that of non-interference with the program under study. For instance, the system has been used to observe in execution portions of the XDS-9300 operating system which previously were not observable because the area in which they reside is also used by all of the other debugging aids available. Students who have had sufficient experience in the XDS assembly language to make use of the system in its present form have found that it is flexible and easy to learn and use.

Although the system developed is useful as it presently stands, several improvements and extensions are possible. As the system is implemented, it is of most use to a programmer engaged in system design, or to a student doing work involving assembly language programming. The display of variable names and decimal values is probably the only function which is especially useful to the novice FORTRAN programmer. The system would be of more value to him if he were also able to access the values by variable name and change them without having to specify absolute memory locations and to input new contents in octal. This could be accomplished if a routine were written which would receive a variable

name through the teletype, scan the NAME LIST for a match, and access the address associated with the name.

One obvious extension which would be of value to any user would be to include relative as well as absolute addressing in the memory block display. A straight-forward method to accomplish this would be to access the symbol-table which is resident at execution time and assign display addresses relative to the nearest symbol-table member for each word in the display block. Some coding has been done on this extension but it is not yet in usable form.

Another, and possibly more valuable, extension would be to provide the user with more control over the execution of the program resident in the XDS. This could be done by using an Execute Address instruction available in the META-SYMBOL statement set. When this instruction is encountered, the word pointed to by the address field is executed. By using this instruction it would be possible to allow the user to step through his program or to execute his program in a slow-continuous mode and to observe the variables change values. To implement this the program would have to maintain a location counter and index registers in the ADAGE memory. It would be necessary to simulate all jump and branch instructions and this would require checking the instruction prior to its being executed.

The last extension recommended would be to implement routines which would allow the user to specify break-points or traps within his program in a more straight-forward manner than is presently allowed. In the present system, the user must actually change machine code in his program in order to do this. It would be very useful for the system to provide the means whereby the user could specify break points in relative terms

by specifying subroutine names or entry point names. This, again, could be accomplished by accessing the resident symbol table.

The software interface between the two memories and the conversion routines implemented in this sytem represent the most time-consuming steps in its evolution. It would be possible to build in many directions from these basic routines. It is intended that applications will be found for these routines in systems developed for other uses.

APPENDIX A

INTERFACE STATEMENT SET

TEST CATEGORY

OPIO 43040 - REQUEST USAGE OF THE INTERFACE

OPIO 43140 - IS THERE A REQUEST PENDING?

OPIO 43120 - IS THIS AGT CONNECTED?

OPIO 43110 - IS THIS AGT NOT PRESENTLY TRANSFERRING?

OPIO 43104 - IS THIS AGT NUMBER ONE?

OPIO 43102 - IS THE XDS PRIORITY INTERRUPT SYSTEM ENABLED?

ACTION CATEGORY

OPIO 43020 - CONTINUE USAGE (MAKE A TRANSFER)

OPIO 43010 - RELEASE THE INTERFACE.

OPIO 43004 - ENABLE THE XDS INTERRUPT SYSTEM

OPIO 43002 - DISABLE THE XDS INTERRUPT SYSTEM

OPIO 43001 - INTERRUPT THE XDS 9300.

APPENDIX B

CONTROL CHARACTERS

7 BIT CODE (IN OCTAL)	OPERATION
000	NULL (NO ACTION)
010	ITALICS (ON OR OFF)
011	POSITION ON X AXIS FOLLOWS
013	POSITION ON Y AXIS FOLLOWS
012	LINE FEED
015	CARRIAGE RETURN
021	DISPLAY VERTICAL (ON OR OFF)
022	DECREASE CHARACTER SIZE
023	INCREASE CHARACTER SIZE
034	START A SUBSCRIPT
035	END SUBSCRIPTING
036	DECREASE BRIGHTNESS
037	INCREASE BRIGHTNESS

```
1.1                     EXPUNGE                         [CLEAR MEM0RY
1.2                     TITLE           DISP
1.3                     ENTRY           DISP,GET10,C0D2,TEMP,GPVT1,PBUF,TEMP
1.4                     ENTRY           GPVT2,ST0RE,PPVT1,PPVT2,BUF,GBUF,FN3,L09K,SPLT
1.5                     ENTRY           EFS,EFG,CVT,ACNVT
1.6                     ENTRY           N1,N2,N3,N4
1.7                     ENTRY           SV1,SV2,SV3,SV4
1.10                    ENTRY           IF
1.11                    ENTRY           Y
1.12                    ENTRY           NP,CC,NN
1.13    SEE:            0
1.14    MIC:            00040JH
1.15    CP:             2124                            [8CPCAL IN 9300
1.16    L90K:           05032JH         [START 9F DISPLAY BJFFER
1.17                    05032JH
1.20                    05450JH
1.21                    11000JH
1.22    WRTAB:          0
1.23    ASCII           $123456789$
1.24                    REPEAT          1000
1.25                    0
1.26                    ENDR
1.27                    1
1.30                    1JH
1.31    GBUF:           REPEAT          10              [BUFFER F0R INPUT FR0M 9300
1.32                    1234567
1.33                    ENDR
1.34                    1JH
1.35    PBUF:           02032100
1.36                    02032002
1.37                    01602124        [GET ENTRY
1.40                    07676551        [ST0RE IT
1.41                    41604560        [GET RNAME
1.42                    07676550        [ST0RE IT
1.43                    176546          [TRAP HERE
```

```
2.1                     REPEAT          10
2.2                     0
2.3                     ENDR
2.4         PPVT1:      0030076540      [PUT PIVOT FOR THE INTERFACE
2.5         PPVT2:      10JH'PBUF       [PUT PIVOT FOR THE INTERFACE
2.6         GPVT1:      76550           [GET PIVOT FOR THE INTERFACE
2.7         GPVT2:      10JH'GBUF       [GET PIV FOR THE INTERFACE
2.10        CMASK:      01400JH         [TURN ON THE CLOCK
2.11        LMASK:      0004000010      [TURN ON THE CHAR GEN
2.12        MD10=30000JH  [DECLARE SOME INSTRUCTIONS
2.13        MD13=33000JH
2.14        MACRO1      BUTON(OLD,FN)   [MACRO TO TEST FUNCTION SWITCHES
2.15                    MDAR'L
2.16                    7777777000
2.17                    MDAS'L
2.20                    1
2.21                    JPAN            .-2             [WAIT
2.22                    MDAR            FNI
2.23                    ARLS            1
2.24                    ARMD            FNI
2.25                    MDBR            OLD
2.26                    ARMD            OLD
2.27                    JPAN            .+2
2.30                    JUMP            .+4
2.31                    BRAR
2.32                    JPAN            .+2
2.33                    JPSR            FN
2.34        ENDM
2.35        FNI:        0               [VARIABLES USED IN BUTON
2.36        OLD1:       0
2.37        OLD2:       0
2.40        OLD3:       0
2.41        OLD4:       0
2.42        OLD5:       0
2.43        OLD6:       0
```

```
3.1          BLD7:          0
3.2          BLD8:          0
3.3          BLD9:          0
3.4          DISP:          NOOP              [START MAIN PROGRAM
3.5                         BPIB              43104          [WHO AM I
3.6                         JPSR              AGT2           [I AM AGT2 GO PATCH
3.7                         MDAR'L            [LOAD THE START OF DISPLAY LIST
3.10                        LOOK
3.11                        ARMD              77735          [STORE IT
3.12                        MDAR'L            [LOAD EOL FOR AVG1
3.13                        RTN
3.14                        ARMD              77736          [SET THE EOL PIVOTS
3.15                        ARMD              77737
3.16                        MDAR'L            [LOAD CLOCK HANDLER PIVOT
3.17                        CLOCK
3.20                        ARMD              77755          [SET THE PIVOT
3.21                        MD10              CMASK
3.22                        JPSR              PUT            [PUT SOME CODE IN 9300
3.23                        BPIB              43001          [INTERRUPT 9300
3.24                        REPEAT            100
3.25                        NOOP
3.26                        NOOP
3.27                        NOOP
3.30                        ENDR
3.31                        JPSR              GET
3.32                        MDAR              GBUF
3.33                        ARMD              GB
3.34                        JPSR              FN3            [GO GET PAGE 1
3.35                        MDAR'L                           [CLEAR THE A REGISTER
3.36                        0
3.37         TOP:           NOOP
3.40                        MDIC'O            MIC
3.41                        SSAR                             [LOAD          BUTTONS
3.42                        ARRS              1
3.43                        ARMD              FNI            [STORE THEM OFF
```

```
4.1                     BUTON(BLD1,FN1)         ;CHECK EACH BUTTON FOR CHANGE
4.2                     BUTON(BLD2,FN2)         ;BRANCH TO SUBROUTINE IF SO
4.3                     BUTON(BLD4,FN4)
4.4                     BUTON(BLD5,FN5)
4.5                     BUTON(BLD6,FN6)
4.6                     BUTON(BLD7,FN7)
4.7                     BUTON(BLD9,FN9)
4.10                    JUMP            TOP     ;LOOP BACK TO TOP
4.11    CLOCK:          NOOP                    ;CLOCK INTERRUPT HANDLER
4.12                    MD10'L                  ;TURN OFF CLOCK
4.13                    400JH
4.14                    MDIC            LMASK   ;TURN ON THE CHARACTER GENERATOR
4.15                    JUMP'I          CLOCK   ;RETURN
4.16    CLK1:           NOOP                    ;DUMMY CLOCK
4.17                    JUMP'I          CLK1
4.20    RMASK:          00040JH
4.21    RTN:            JUMP            .
4.22                    MDAR'L                  ;EOL HANDLER FOR AVG1
4.23                    LOOK                    ;RESET THE HEAD OF LIST
4.24                    ARMD            77735
4.25                    MDIC            RMASK   ;TURN OFF CHAR GEN
4.26                    MD10            CMASK   ;TURN ON THE CLOCK
4.27                    JUMP'I          RTN     ;RETURN
4.30    GET:            NOOP                    ;GET SOME WORDS FROM 9300
4.31                    MDAR'L                  ;LOAD THE PIVS
4.32                    MD13            GPVT1
4.33                    ARMD            77730
4.34                    MDAR'L
4.35                    MD13            GPVT2
4.36                    ARMD            77731
4.37                    JUMP            .+4     ;JUMP AROUND SOME PUT CODE
4.40    WRT:            NOOP
4.41                    MDAR            WRT     ;RESET RETURN FOR A PUT
4.42                    ARMD            GET
4.43                    FPRI                    ;FREEZE INTERRUPTS
```

```
5.1                     SPIO            43102           [SDS PRI ENABLED
5.2                     SPIO            43004           [ENABLE IT
5.3                     SPIO            43120           [AM I CONNECTED
5.4                     JPSR            RQUSE           [GO GET CONNECTED
5.5                     SPIO            43020           [CONTINUE USAGE OF CHANNEL
5.6                     UPRI                            [UNFREEZE PRIS
5.7                     SPIO            43110           [AM I FINISHED
5.10                    JUMP            .-1             [WAIT FOR FINISH
5.11                    SPIO            43010           [RELEASE INTERFACE
5.12                    JUMP'I          GET             [RETURN
5.13    AGT2:           JUMP            .               [MODIFY THE INTERRUPTS
5.14                    MOAR'L                          [LOAD AN AGT2 INTERRUPT
5.15                    00232010
5.16                    ARMD            XINT1           [STORE IT
5.17                    ARMD            XINT2           [AND AGAIN
5.20                    JUMP'I          AGT2            [RETURN
5.21    BUF:            0                               [MARK PRES ADDR IN 9300
5.22    FN9:            JUMP            .
5.23                    JPSR            $OFST           [OUTPUT ON TTY
5.24    NOCARRET
5.25    STRING '
5.26    ENTER ADDRESS='
5.27                    CARRET
5.30                    JPSR            $ICHTY          [INPUT A DIGIT
5.31                    3                               [ECHO OUT THE CHAR
5.32                    MOAR'L'A                        [MASK IT
5.33                    7
5.34                    ARLS            12.             [SHIFT IT
5.35                    NOOP
5.36                    NOOP
5.37                    ARBR
5.40                    JPSR            $ICHTY          [GET ANOTHER
5.41                    3
5.42                    MOAR'L'A                        [MASK   SHIFT
5.43                    7
```

```
6.1                     ARLS            9.
6.2                     N90P
6.3                     N90P
6.4                     AR2R'0
6.5                     JPSR            $ICHTY          [AGAIN
6.6                     3
6.7                     MDAR'L'A
6.10                    7
6.11                    ARLS            6
6.12                    N90P
6.13                    N90P
6.14                    AR2R'0
6.15                    JPSR            $ICHTY          [AND.....
6.16                    3
6.17                    MDAR'L'A
6.20                    7
6.21                    ARLS            3
6.22                    N90P
6.23                    AR8R'0
6.24                    JPSR            $ICHTY          [AGAIN
6.25                    3
6.26                    MDAR'L'A
6.27                    7
6.30                    AR8R'0
6.31                    BRMD            SV4             [SAVE NR
6.32                    JPSR            $0FST           [WRITE A STRING
6.33        N0CARRET
6.34        STRING '
6.35        NEW CONTENTS= '
6.36        CARRET
6.37                    JPSR            $ICHTY          [GET...
6.40                    3
6.41                    ARLS            21.             [AND....
6.42                    N90P                            [MASK.....
6.43                    N90P                            [NEW
```

```
7.1                     ARBR                            [CONTENTS
7.2                     JPSR            $ICHTY
7.3                     3
7.4                     MDAR'L'A
7.5                     7
7.6                     ARLS            18.
7.7                     ARBR'0
7.10                    JPSR            $ICHTY
7.11                    3
7.12                    MDAR'L'A
7.13                    7
7.14                    ARLS            15.
7.15                    ARBR'0
7.16                    JPSR            $ICHTY
7.17                    3
7.20                    MDAR'L'A
7.21                    7
7.22                    ARLS            12.
7.23                    ARBR'0
7.24                    JPSR            $ICHTY
7.25                    3
7.26                    MDAR'L'A
7.27                    7
7.30                    ARLS            9.
7.31                    ARBR'0
7.32                    JPSR            $ICHTY          [AND STORE IT.....
7.33                    3                               [INTO BREG
7.34                    MDAR'L'A
7.35                    7
7.36                    ARLS            6
7.37                    NOOP
7.40                    NOOP
7.41                    ARBR'0
7.42                    JPSR            $ICHTY
7.43                    3
```

```
10•1                    MDAR'L'A
10•2                    7
10•3                    ARLS            3
10•4                    ARBR'0
10•5                    JPSR            $ICHTY
10•6                    3
10•7                    MDAR'L'A
10•10                   7
10•11                   ARBR'0
10•12                   BRAR                        [LOAD A
10•13                   MDAR'L'A                    [MASK OFF GARBAGE
10•14                   77777777
10•15                   ARBR                        [MAKE UP THEXFER PIVS
10•16                   MDAR            PPVT1
10•17                   ARMD            SV1
10•20                   MDAR            PPVT2
10•21                   ARMD            SV2
10•22                   MDAR            SV4
10•23                   MDAR'L'0
10•24                   30JVH
10•25                   ARMD            PPVT1       [LOAD THE XFER PIVOTS
10•26                   BRMD            SV3
10•27                   MDAR'L
10•30                   1JVH'SV3
10•31                   ARMD            PPVT2
10•32                   JPSR            PUT         [STUFF IT IN 93MEM
10•33                   MDAR            SV1
10•34                   ARMD            PPVT1
10•35                   MDAR            SV2
10•36                   ARMD            PPVT2
10•37                   MDAR'L                      [SET THE CLOCK
10•40                   CLOCK
10•41                   ARMD            77755
10•42                   MDAR'L
10•43                   RTN
```

```
11•1                    ARMD        77736
11•2                    ARMD        77737
11•3                    JPSR        FN1         [G0 WAIT F0R A J0B
11•4                    JUMP'I      FN9         [G0 H9ME
11•5        SV1:        0
11•6        SV2:        0
11•7        SV3:        0
11•10       SV4:        0
11•11       CVT:        JJMP        •           [BCD/ASCII C0NVERSI0N
11•12                   ARMD        ESA         [ST0RE BCD
11•13                   MDAR'L'A                [AND 0FF 0NE
11•14                   7777
11•15                   JPSR        ACNVT       [G0 C0NVERT IT
11•16                   ARMD        EST         [SAVE THE ASCII
11•17                   MDAR        ESA         [GET S0ME M0RE BCD
11•20                   ARRS        12•         [P0SIT F0R A CALL
11•21                   N00P
11•22                   N00P
11•23                   JPSR        ACNVT       [G0 D0 IT
11•24                   ARLS        15•         [REP0SITI0N IT
11•25                   N00P
11•26                   N00P
11•27                   ARBR
11•30                   MDAR        EST         [L0AD REGS F0R RETURN
11•31                   ARBR'0
11•32                   BRAR
11•33                   JUMP'I      CVT         [G0 H9ME
11•34       ACNVT:      JUMP        •           [D0 THE C0NVERSI0N
11•35                   ARMD        EFS         [ST0RE BCD HALFWD
11•36                   MDAR'L'A                [GET 0NE CHARACTER
11•37                   77
11•40                   MDAE'L                  [ADD 0N TABLE T0P
11•41                   BCDTB
11•42                   ARMD        EFG         [SAVE THIS INDEX
11•43                   MDAR'I      EFG         [L0AD THE ASCII CHAR
```

```
12.1                    ARLS        1           [PØSIT FØR DISPLAY
12.2                    ARBR                    [SAVE IT IN BREG
12.3                    MDAR        EFS         [LØAD NEXT CHAR
12.4                    ARRS        6
12.5                    NØØP
12.6                    NØØP
12.7                    MDAR'A'L                [MAKE
12.10                   77                      [AN INDEX
12.11                   MDAE'L
12.12                   BCDTB
12.13                   ARMD        EFG         [STØRE INDEX
12.14                   MDAR'I      EFG         [LØAD WØRD INDIRECT
12.15                   ARLS        8.          [PØSITIØN IT
12.16                   NØØP
12.17                   NØØP
12.20                   ARBR'Ø      [ØR IT IN
12.21                   BRAR                    [LØAD ASCII HALFWD
12.22                   JUMP'I      ACNVT       [RETURN
12.23       EFS:        0                       [WØRK SPACE.....
12.24       EFG:        0
12.25       ESA:        0
12.26       EST:        0
12.27       RBAST:      2075
12.30       RNAME:      4560
12.31       NAM2P:      0
12.32       NAM1P:      0
12.33       LØCP:       0
12.34       BCDTB:      60
12.35                   61
12.36                   62
12.37                   63
12.40                   64
12.41                   65
12.42                   66
12.43                   67
```

DATE

13.1	70
13.2	71
13.3	60
13.4	75
13.5	47
13.6	72
13.7	76
13.10	133
13.11	53
13.12	101
13.13	102
13.14	103
13.15	104
13.16	105
13.17	106
13.20	107
13.21	110
13.22	111
13.23	77
13.24	56
13.25	51
13.26	133
13.27	134
13.30	43
13.31	55
13.32	112
13.33	113
13.34	114
13.35	115
13.36	116
13.37	117
[illegible]	120
[illegible]	121
[illegible]	122
[illegible]	41

```
14.1                        44
14.2                        52
14.3                        135
14.4                        73
14.5                        44
14.6                        40
14.7                        57
14.10                       123
14.11                       124
14.12                       125
14.13                       126
14.14                       127
14.15                       130
14.16                       131
14.17                       132
14.20                       040
14.21                       54
14.22                       50
14.23                       100
14.24                       57
14.25                       77
14.26   NAM1:               0
14.27   NAM2:               0
14.30   NEG:                JUMP            .               [CONVERT A NEG TO POS
14.31                       ARAR'N                          [NEGATE IT
14.32                       ARRS            7               [POSITION IT
14.33                       NOOP
14.34                       NOSP
14.35                       ARBR
14.36                       MQAR'L
14.37                       1
14.40                       ARMQ            MINUS           [SAVE IT OFF
14.41                       BRAR                            [LOAD ANS
14.42                       JUMP'I          NEG             [RETURN
14.43   MINUS:              0
```

```
15•1        RQUSE:      JUMP        •               [REQ USE OF INTERFACE
15•2                    OPIO        43040           [REQ USAGE
15•3                    JUMP        •-1             [WAIT FOR A ROGER
15•4                    OPIO        43120           [AM I CONNECTED YET
15•5                    JUMP        •-1             [NO•••WAIT
15•6                    JUMP'I      RQUSE           [YES•••GO HOME
15•7        PUT:        NOOP                        [PUT CODE INTO 9300
15•10                   MDAR'L                      [LOAD PIVS FOR A PUT
15•11                   MD13        PPVT1
15•12                   ARMD        77730
15•13                   MDAR'L
15•14                   MD13        PPVT2
15•15                   ARMD        77731
15•16                   JPSR        WRT             [GO UP AND USE CODE IN GET
15•17                   JUMP'I      PUT             [RETURN
15•20       GET10:      NOOP                        [GET THE PROGRAM LOCATION IN9300
15•21                   MDAR'L                      [MODIFY THE PUT PIVOTS
15•22                   11JH'COD2
15•23                   ARMD        PPVT2
15•24                   JPSR        PUT             [STUFF IN THE CODE
15•25                   JPSR        GET             [GO GET THE PROG LOC
15•26                   JUMP'I      GET10           [RETURN
15•27       COD2:       07777770
15•30                   11776547                    [LOAD INDEX REGISTER
15•31                   11602135
15•32                   17676560                    [GET THE WORD AND STORE IT
15•33                   15776542                    [LOOP 8 TIMES
15•34       XINT1:      00232020                    [INTERRUPT ADAGE WHEN DONE
15•35                   00176546
15•36                   00177770                    [-8 FOR X1
15•37       CODIT:      11600010                    [9300 COMMAND FOR LATER GETS
15•40       FN2:        NOOP                        [DISPLAY NAMELIST
15•41                   MDAR'L                      [RESET THE DISPLAY POINTER
15•42                   WRTAB
15•43                   ARMD        LPT
```

```
16•1                MDAR            GB              [HEAD 0F NAMELIST
16•2        W:      MDAR'L'A        [MASK 0FF THE GARBAGE
16•3                77777
16•4                ARMD            NW              [L0AD A P0INTER
16•5                ARMD            GPVT1           [SET THE GETPIVS
16•6                MDAR'L          [TURN 0FF THE CHAR GEN••
16•7                CLK1            [IT SCREWS UP THE INT
16•10               ARMD            77755
16•11               JPSR            RTN             [CLEAN UP CHAR GEN
16•12               JPSR            GET             [G0 GET THE NAMELIST HEADER
16•13               MDAR            GBUF
16•14               JPLS            •+11
16•15               MDAR'L                          [IF S0 THEN
16•16               76550                           [RESET
16•17               ARMD            GPVT1           [THE PIV
16•20               MDAR'L                          [ADD AN E9L FLG BIT
16•21               1'H
16•22               ARMD'X'I        LPT             [INT0 THE DISPLAY
16•23               JPSR            FN1             [HANG UP A WHILE
16•24               JUMP'I          FN2             [AND G0 H9ME
16•25               ARMD            NP              [IF N0 L0AD NR IN LIST
16•26               MDAR            GBUF+1          [AND MAKE A NEG P0INTER
16•27               ARAR'N
16•30               MDAE'L          [ADD 0NE F0R TW0'S C0MPLEMENT
16•31               1
16•32               ARMD            CC              [L0AD A C0UNTER F0R L00P
16•33               MDAR            LNFD
16•34               ARMD'X'I        LPT
16•35               MDAR            NW              [L0AD PTR
16•36               MDAS'L          [BUMP IT
16•37               2
16•40               ARMD            NW
16•41       XNX:    ARMD            GPVT1           [SET GETPIV
16•42               JPSR            GET             [GET IST ENTRY
16•43               MDAR            GBUF
```

```
17.1                    JPSR        CVT         [CONVERT TO ASCII
17.2                    ARMD'X'I    LPT         [DISPLAY IT
17.3                    MDAR        GBUF+1
17.4                    JPSR        CVT
17.5                    ARMD'X'I    LPT
17.6                    MDAR        GBUF+2      [GET POINTER TO VALUE
17.7                    ARLS        13.
17.10                   NOOP
17.11                   JSAN        REAL        [GO FIX NUMBER
17.12                   MDAR        GBUF+2      [GET THE POINTER
17.13                   MDAR'L'A    [CLEAN IT UP
17.14                   77777
17.15                   ARMD        GPVT1       [LOAD THE PIVOTS
17.16                   MDAR'L
17.17                   100
17.20                   ARMD'X'I    LPT         [STORE A SPACE IN DISPLAY
17.21                   JPSR        GET         [GET THE VALUE
17.22                   MDAR        GBUF
17.23                   JPSR        CDEC        [CONVERT OCTAL TO DEC
17.24                   ARMD'X'I    LPT         [STORE IT IN DISPLAY
17.25                   BRMD'X'I    LPT
17.26                   MDAR        LNFD        [STORE A LINE FEED
17.27                   ARMD'X'I    LPT
17.30       CX:         MDAR        CC          [LOAD AND BUMP
17.31                   MDAE'L
17.32                   1
17.33                   ARMD        CC
17.34                   JPAN        .+5         [TEST COUNTER
17.35                   MDAR        NP          [LOAD ANOTHER
17.36                   JPLS        W           [TEST IF MORE
17.37                   JPSR        FN1         [GO HOME IF NOT
17.40                   JUMP'I      FN2
17.41                   MDAR        NW          [CONTINUE IF SO
17.42                   MDAE'L                  [BUMP INSIDE CTR
17.43                   3
```

```
20•1                    ARMD            NW
20•2                    JUMP            XNX             [L00P BACK
20•3        FN1:        N00P                            [TRAP EXECUTI0N
20•4                    JPSR            RTN             [CLEAN UP DISPLAY
20•5                    MDAR'L                          [RESET CL0CK
20•6                    CL0CK
20•7                    ARMD            77755
20•10                   MDIC'0          MIC             [GET THE BUTT0NS
20•11                   S5AR
20•12                   JPAN            •+2             [WAIT F0R NUMBER 0NE
20•13                   JUMP            •-3
20•14                   JUMP'I          FN1             [RETURN WHEN SENSED
20•15       CC:         0               [ST0RAGE F0R FNS
20•16       NP:         0
20•17       NW:         0
20•20       LPT:        0
20•21       E:          212             [AN ASCII E
20•22       SSS:        0
20•23       REAL:       N00P            [C0NVERT REAL T0 FIXED
20•24                   ARRS            13•
20•25                   N00P
20•26                   N00P
20•27                   ARMD            SSS
20•30                   MDAR'L'A        [SHIFT AND MASK
20•31                   77777
20•32                   ARMD            GPVT1           [SET THE PIV0T
20•33                   JPSR            GET             [GET THE EXP0NENT
20•34                   MDAR            GBUF+1          [FIX THE NUMBER
20•35                   ARAR'N
20•36                   AR0R                            [USE XP0N F0R CTR
20•37                   MDAR            GBUF            [GET MANT•
20•40                   ARRS            9•              [P0SITI0N IT
20•41                   N00P
20•42                   N00P
20•43                   ARMD            IF              [SAVE IT
```

```
21•1        XXQ:        0
21•2                    BRAR
21•3                    MDAS'L                          [C0UNT SHIFTS
21•4                    1
21•5                    ARBR
21•6                    JPAN        XQ                  [AN0THER SHIFT
21•7                    MDAR        IF                  [SAVE IT
21•10                   ARRS        15.                 [SHIFT IT
21•11                   JUMP        CXX                 [SET F0R RTN
21•12       XQ:         MDAR        IF                  [L0AD W0RD
21•13                   ARLS        1                   [MPY BY 2
21•14                   ARMD        IF                  [ST0RE IT
21•15                   JUMP        XXG                 [G0 BACK AGAIN
21•16       CXX:        N00P
21•17                   JPSR        CDEC                [G0 C0NVERT
21•20                   ARMD'X'I    LPT                 [ST0RE
21•21                   BRMD'X'I    LPT                 [AND ST0RE
21•22                   MDAR        E                   [L0AD AN E
21•23                   ARMD'X'I    LPT                 [ST0RE IT
21•24                   MDAR'L                          [L0AD XP0N
21•25                   1
21•26                   JPSR        CDEC                [C0NVERT IT
21•27                   ARMD'X'I    LPT                 [ST0RE IT
21•30                   BRMD'X'I    LPT
21•31                   JUMP        CX-2                [G0 H0ME
21•32       IF:         0
21•33       CDEC:       JUMP        .                   [C0NVERT 0CTAL T0 DECIMAL
21•34                   N00P
21•35                   N00P
21•36                   N00P
21•37                   MDAR'L'A    [CLEAN IT UP
21•40                   7777
21•41                   ARMD        Y
21•42                   MDAR'L
21•43                   0
```

```
22.1                ARMD            N1
22.2                ARMD            N2
22.3                MDAR            Y
22.4                DIVI            5000.           [GET THE FIRST DIGIT
22.5                N88P
22.6                N88P
22.7                N88P
22.10               ARMD            N3              [SAVE IT
22.11               MPYI            5000.           [GET THE REMAINDER
22.12               N88P
22.13               N88P
22.14               ARBR'N
22.15               MDAR            Y
22.16               BRAS
22.17               ARMD            Y
22.20               DIVI            500.            [GET THE SECBND DIGIT
22.21               N88P
22.22               N88P
22.23               N88P
22.24               ARMD            N4              [SAVE IT
22.25               MPYI            500.
22.26               N88P
22.27               N88P
22.30               ARBR'N
22.31               MDAR            Y
22.32               BRAS                            [GET THE REST BF..
22.33               ARMD            Y               [THE DIGITS AND..
22.34               DIVI            50.             [SAVE THEM IN THE...
22.35               N88P                            [FBRM BF INDICIES
22.36               N88P                            [INTB AN
22.37               ARMD            N5              [ASCII TABLE
22.40               MPYI            50.             [WHICH CBNTAINS 0-9
22.41               N88P
22.42               N88P
22.43               ARBR'N
```

```
23.1        MDAR        Y
23.2        BRAS
23.3        ARMD        Y
23.4        DIVI        5.
23.5        N88P
23.6        N88P
23.7        ARMD        N6
23.10       MPYI        5
23.11       N99P
23.12       N99P
23.13       ARBR'N
23.14       MDAR        Y
23.15       BRAS
23.16       ARMD        Y
23.17       MDAE'L                  [INDEX INTO TABLE
23.20       LST+1
23.21       ARMD        LST
23.22       MDAR'I      LST         [LOAD THE ASCII CHAR
23.23       ARMD        N7          [SAVE IT
23.24       MDAR'L
23.25       LST+1
23.26       MDAS        N6          [DO IT AGAIN FOR NEXT DIGIT
23.27       ARMD        LST
23.30       MDAR'I      LST
23.31       ARMD        N6          [AND AGAIN
23.32       MDAR'L
23.33       LST+1                   [THIS
23.34       MDAS        N5          [IS DONE
23.35       ARMD        LST         [FOR ALL OF THE
23.36       MDAR'I      LST         [DECIMAL DIGITS
23.37       ARMD        N5          [IN THE WORD
24.40       MDAR'L
24.41       LST+1
24.42       MDAS        N4          [FOURTH DIGIT
24.43       ARMD        LST         [MAKE INDEX
```

```
24.1            MDAR'I          LST
24.2            ARMD            N4              [STORE ASCII
24.3            MDAR'L
24.4            LST+1
24.5            MDAS            N3              [THIRD DIGIT
24.6            ARMD            LST
24.7            MDAR'I          LST
24.10           ARMD            N3
24.11           MDAR'L
24.12           LST+1
24.13           MDAS            N2              [SECOND DIGIT
24.14           ARMD            LST
24.15           MDAR'I          LST
24.16           ARMD            N2
24.17           MDAR'L
24.20           LST+1
24.21           MDAS            N1              [FIRST DIGIT
24.22           ARMD            LST
24.23           MDAR'I          LST
24.24           ARMD            N1
24.25           ARLS            23.             [FORMAT THE
24.26           NOOP                            [ASCII CHARACTERS INTO
24.27           NOOP                            [DISPLAY WORDS
24.30           ARBR
24.31           MDAR            N2
24.32           ARLS            16.
24.33           NOOP
24.34           NOOP
24.35           ARBR'0                          [AND STORE IN BREG
24.36           MDAR            N3
24.37           ARLS            8.
24.40           NOOP
24.41           NOOP
24.42           ARBR'0
24.43           MDAR            N4
```

```
25.1                    ARLS            1
25.2                    ARBR'0
25.3                    BRAR
25.4                    MDAR'L'A                        [TURN 0FF
25.5                    7777677776                      [E0L FLAGS
25.6                    ARMD            LST
25.7                    MDAR            N5              [MAKE AN0THER W0RD
25.10                   ARLS            16.             [WITH THE REST 0F THE DIGITS
25.11                   N00P
25.12                   N00P
25.13                   ARBR
25.14                   MDAR            N6
25.15                   ARLS            8.
25.16                   N00P
25.17                   ARBR'0
25.20                   MDAR            N7
25.21                   ARLS            1
25.22                   ARBR'0
25.23                   MDAR            LST             [L0AD A WITH FIRST B HAS 2
25.24                   JUMP'I          CDEC            [RETURN
25.25   FN3:            N00P            [GET THE FIRST PAGE 0F C0DE
25.26                   MDAR            CP              [L0AD &CPCAL
25.27                   MDAE            C0DIT           [ADD 0N THE INSTRUCTI0N
25.30                   ARMD            C0D2+2          [L0AD INT0 C0DE F0R 9300
25.31                   JPSR            GET10           [00 GET THE START ADDRESS
25.32                   MDAR            GBUF+1
25.33   M0:             MDAR'L'A        [CLEAR 0FF GARBAGE
25.34                   77777
25.35                   ARMD            BUF             [PUT IT IN A P0INTER
25.36                   ARMD            P               [L0AD C0UNTER F0R LINE NRS
25.37                   MDAR'L
25.40                   CLK1
25.41                   ARMD            77755
25.42                   JPSR            RTN
25.43                   MDAR'L                          [SET DISPLAY P0INTER
```

```
26.1                    WRTAB
26.2                    ARMD            WRPT
26.3                    MDAR'L                          [SET A L00P C0UNTER
26.4                    -4
26.5                    ARMD            CT2
26.6            M1:     MDAR            C0DIT           [SET UP T0 GET 8 WDS 0F 93C0DE
26.7                    MDAE            BUF
26.10                   MDAR'L'A
26.11                   11677777
26.12                   ARMD            C0D2+2          [M0DIFY 93INSTS
26.13                   MDAR'L
26.14                   00176540
26.15                   ARMD            C0D2+6
26.16                   MDAR'L
26.17           XINT2:  00232020
26.20                   ARMD            C0D2+5
26.21                   MDAR'L                          [SET UP F0R INTERRUPT FM 9300
26.22                   HND
26.23                   ARMD            77732
26.24                   JPSR            PUT             [L0AD IN THE C0DE
26.25                   JUMP            .               [WAIT F0R AN INTERRUPT
26.26           HND:    N00P            [RETURN HERE AFTER INT
26.27                   FPRI            [TURN 0FF THE INTERRUPTS
26.30                   MDAR'L          [0VERWRITE INTERRUPT C0DE IN 9300
26.31                   01000000        [93 N0P
26.32                   ARMD            C0D2+5          [GET READY T0 SEND IT
26.33                   JPSR            PUT             [PUT IN 9300
26.34                   JPSR            GET             [G0 GET EIGHT W0RDS
26.35           NXT:    N00P
26.36                   MDAR'L          [SET INDEX INT0 C0DE IN GBUF
26.37                   GBUF-1
26.40                   ARMD            GPT
26.41                   MDAR'L          [SET A 10 C0UNTER
26.42                   -7
26.43                   ARMD            CT10
```

```
27.1          M2:     MDAR'I'X        GPT             [GET THE FIRST WORD
27.2                  JPSR            BRKIT           [GO CONVERT IT FOR DISPLAY
27.3                  MDAR'L                          [LOAD A COUNTER
27.4                  -10
27.5                  ARMD            CT5
27.6                  MDAR'L          [SET ANOTHER COUNTER
27.7                  STORE-4
27.10                 ARMD            STCNT
27.11         M3:     MDAR'X'I        STCNT           [GET THE META SYMBOL
27.12                 MDAR'L'A                        [CLEAR EOL FLAG
27.13                 7777677776
27.14                 ARMD'X'I        WRPT            [STORE THE META IN DISPLAY
27.15                 MDAR'L          [INDEX A COUNTER
27.16                 1
27.17                 MDAE            CT5
27.20                 ARMD            CT5
27.21                 JPAN            M3              [LOOP IF NOT FINISHED
27.22         HHH:    NOOP
27.23                 MDAR            LNFD            [LINE FEED INTO DISPLAY
27.24                 ARMD'X'I        WRPT
27.25                 MDAR'L
27.26                 1
27.27                 MDAE            CT1C            [TEST TO SEE IF DONE
27.30                 ARMD            CT1C
27.31                 JPAN            M2              [LOOP IF NOT
27.32                 MDAR            BUF
27.33                 MDAS'L                          [GET SET FOR NEXT 8 WORDS
27.34                 10
27.35                 ARMD            BUF             [LOAD THE LOC COUNTER
27.36                 MDAR'L
27.37                 1
27.40                 MDAE            CT2             [SEE IF THATS ALL
27.41                 ARMD            CT2
27.42                 JPAN            M1              [JUMP IF NOT
27.43                 MDAR            END             [LOAD AN EOL
```

```
30.1                    ARMD'X'I        WRPT            [ST0RE IT
30.2                    MDAR'L
30.3                    CL0CK
30.4                    ARMD            77755
30.5                    JUMP'I          FN3             [G0 H9ME
30.6        BRKIT:      N00P            [BREAK THE 93W0RD INT0 DISPLAY F0RM
30.7                    ARMD            TEMP            [GET THE W0RD
30.10                   ARRS            17              [SHIFT T0 0PC0DE
30.11                   N00P
30.12                   N00P
30.13                   MDAR'L'A        [MASK REST
30.14                   77
30.15                   MDAS'L          [MAKE AN INDEX
30.16                   INST
30.17                   ARMD            TINS
30.20                   MDAR'I          TINS            [L0AD THE 0PC0DE
30.21                   MDAR'L'A        [RESET E0L FLAG
30.22                   7777777776
30.23                   ARMD            ST0RE           [SAVE IT
30.24                   MDAR            TEMP            [GET THE W0RD
30.25                   MDAR'L'A        [RESET ALL BUT ADDR FIELD
30.26                   77777
30.27                   JPSR            SPLT            [G0 F0RMAT IT
30.30                   MDAR'L'A        [CLEAN IT UP
30.31                   777777
30.32                   ARMD            ST0RE+1         [ST0RE THE ASCII
30.33                   BRMD            ST0RE+2
30.34                   MDAR'L          [L0AD A SPACE
30.35                   100             [ST0RE IT
30.36                   ARMD            ST0RE+3
30.37                   MDAR            TEMP
30.40                   JPSR            SPLT            [GET WH0LE WD
30.41                   ARMD            ST0RE+4         [C0NVERT AND ST0RE
30.42                   BRMD            ST0RE+5
30.43                   MDAR            P               [INSERT THE
```

```
31.1                    JPSR            SPLT            [WORD AFTER BREAKING
31.2                    MDAR'L'A                        [ALL OF IT
31.3                    156                             [THIS TIME WITH
31.4                    ARMD            NR              [CONTROL CHARS AND
31.5                    BRMD            NR+1            [LOC NUMBER
31.6                    MDAR'L
31.7                    1
31.10                   MDAE            P
31.11                   ARMD            P
31.12                   JUMP'I          BRKIT           [RETURN
31.13           P:      0
31.14           SPLT:   NOOP            [CONVERT A HALF WORD TO ASCII
31.15                   ARMD            S               [THIS IS DONE
31.16                   JPSR            SP1             [WITH A SERIES OF SHIFTS AND MASKS
31.17                   BRMD            SL              [TO GET THE 8 DIGITS
31.20                   MDAR            S               [INTO A FORM FOR DISPLAY
31.21                   ARRS            12.
31.22                   NOOP
31.23                   NOOP
31.24                   MDAR'L'A
31.25                   7777
31.26                   JPSR            SP1
31.27                   BRAR
31.30                   MDBR            SL
31.31                   JUMP'I          SPLT
31.32           SL:     0               [SAVE THE VALUE
31.33           SP1:    NOOP            [INSERT THE 60'S
31.34                   ARMD            SV:"            [GET THE HALFWD
31.35                   MDAR'L'8                        [MAKE IT ASCII
31.36                   60
31.37                   ARLS            1               [POSIT FOR DISPLAY
31.40                   MDAR'L'A                        [AND IN A MASK
31.41                   156
31.42                   ARBR            [SAVE IT IN B
31.43                   MDAR            SVIT            [GET ANOTHER
```

```
32•1                    ARRS            3
32•2                    ARMD            SVIT
32•3                    MDAR'L'0
32•4                    60
32•5                    ARLS            8.              [DO IT AGAIN
32•6                    N96P
32•7                    N98P
32•10                   MDAR'L'A
32•11                   33400           [ANOTHER SHIFT AND MASK
32•12                   ARBR'0
32•13                   MDAR            SVIT
32•14                   ARRS            3
32•15                   ARMD            SVIT
32•16                   MDAR'L'0
32•17                   60
32•20                   ARLS            16.
32•21                   N80P
32•22                   N96P
32•23                   MDAR'L'A
32•24                   156JH
32•25                   ARBR'0
32•26                   MDAR            SVIT            [LOAD SHIFT AND MASK
32•27                   ARRS            3
32•30                   ARMD            SVIT
32•31                   MDAR'L'0
32•32                   60
32•33                   ARLS            23.             [SHIFT AND MASK
32•34                   N98P
32•35                   N98P
32•36                   N98P
32•37                   MDAR'L'A        [AND AGAIN
33•40                   33400JH
33•41                   ARBR'0
33•42                   JUMP'I          SP1             [RETURN
33•43           S:      0
```

```
33.1        WRPT:       0
33.2        CT2:        0
33.3        GPT:        0
33.4        CT10:       0
33.5        CT5:        0
33.6        STCNT:      0
33.7        TEMP:       0
33.10       INST:       ASCII       $HLT $      [INSTRUCTIONS IN ASCII
33.11                   ASCII       $BRU $
33.12                   ASCII       $EOM $
33.13                   ASCII       $BRM $
33.14                   ASCII       $SUB $
33.15                   ASCII       $ADD $
33.16                   ASCII       $LDS $
33.17                   ASCII       $INT $
33.20                   ASCII       $NOP $
33.21                   ASCII       $ETR $
33.22                   ASCII       $EOR $
33.23                   ASCII       $MRG $
33.24                   ASCII       $LDB $
33.25                   ASCII       $EAX $
33.26                   ASCII       $LDA $
33.27                   ASCII       $LDX $
33.30                   ASCII       $CIT $
33.31                   ASCII       $EXU $
33.32                   ASCII       $FLC $
33.33                   ASCII       $REP $
33.34                   ASCII       $DPS $
33.35                   ASCII       $DPA $
33.36                   ASCII       $LDP $
33.37                   ASCII       $DPM $
33.40                   ASCII       $MIA $
33.41                   ASCII       $POT $
33.42                   ASCII       $AIN $
33.43                   ASCII       $PIN $
```

DATE

```
34.1    ASCII   $XMB
34.2    ASCII   $ADM
34.3    ASCII   $XMA
34.4    ASCII   $XVX
34.5    ASCII   $RCJ
34.6    ASCII   $BRK
34.7    ASCII   $E9C
34.10   ASCII   $BVA
34.11   ASCII   $SKL
34.12   ASCII   $SKE
34.13   ASCII   $SKG
34.14   ASCII   $SKD
34.15   ASCII   $SKF
34.16   ASCII   $SKE
34.17   ASCII   $SKE
34.20   ASCII   $SKA
34.21   ASCII   $SKA
34.22   ASCII   $SKG
34.23   ASCII   $SKG
34.24   ASCII   $BRX
34.25   ASCII   $SFT
34.26   ASCII   $TYU
34.27   ASCII   $DIV
34.30   ASCII   $MUL
34.31   ASCII   $FLS
34.32   ASCII   $FLK
34.33   ASCII   $FLD
34.34   ASCII   $FLH
34.35   ASCII   $SFS
34.36   ASCII   $MPD
34.37   ASCII   $MP
21.40   ASCII   $SKR
34.41   ASCII   $SFB
34.42   ASCII   $STD
21.43   ASCII   $ST
```

```
35.1                    ASCII           $STZ $
35.2                    ASCII           $STX $
35.3        ASCII       $     $
35.4        PPT:        0               [POINTERS
35.5        NMPT:       0
35.6        GB:         0
35.7        BUFF:       0
35.10       LOCCT:      0
35.11       CTR:        0
35.12       Y:          0               [WORK AREA FOR CONVERSION ROUTINES
35.13       N1:         0
35.14       N2:         0
35.15       N3:         0
35.16       N4:         0
35.17       N5:         0
35.20       N6:         0
35.21       N7:         0
35.22                   0
35.23                   0
35.24       LST:        0                           [ASCII NUMERALS
35.25                   60
35.26                   061
35.27                   62
35.30                   63
35.31                   64
35.32                   65
35.33                   66
35.34                   67
35.35                   70
35.36                   71
35.37       FN7:        NORP            [RESET THE LOC IN 93MEM
35.40                   MDAR'L                      [SET UP TO EXEC FN3
35.41                   R4
35.42                   ARMD            FN3
35.43                   MDAR'L
```

```
36.1                    15
36.2                    MD10'L                          [CLEAN UP CHAR GEN
36.3                    0
36.4                    JPSR            $0FST           [0UTPUT A STRING
36.5        N0CARRET
36.6        STRING '
36.7        START ADDRESS= '
36.10                   CARRET
36.11                   JPSR            $ICHTY          [GET A DIGIT
36.12                   3                               [AND FEED IT BACK
36.13                   MDAR'L'A                        [P0SITI0N IT
36.14                   7                               [MASK IT
36.15                   ARLS            12.
36.16                   N90P
36.17                   N90P
36.20                   N90P
36.21                   AR0R
36.22                   JPSR            $ICHTY          [GET AN0THER DIGIT
36.23                   3
36.24                   MDAR'L'A                        [MASK
36.25                   7
36.26                   ARLS            9.              [BUILD A NUMBER
36.27                   N90P
36.30                   N90P
36.31                   ARBR'0
36.32                   JPSR            $ICHTY          [GET AN0THER 0NE
36.33                   3
36.34                   MDAR'L'A
36.35                   7
36.36                   ARLS            6
36.37                   N90P
36.40                   N90P
36.41                   ARBR'0
36.42                   JPSR            $ICHTY          [NEXT DIGIT
36.43                   3
```

```
37.1                    MDAR'L'A
37.2                    7
37.3                    ARLS            3
37.4                    N90P
37.5                    AR8R'8
37.6                    JPSR            $ICHTY          [LEAST SIGNIFICANT DIGIT IN ADDR
37.7                    3
37.10                   MDAR'L'A
37.11                   7
37.12                   AR5R'8
37.13                   MDAR'L'A
37.14                   77777
37.15                   BRAR
37.16                   ARMD            SEE             [RESET THE P8INTERS
37.17                   MDAR'L
37.20                   RTN
37.21                   ARMD            77735
37.22                   ARMD            77736           [CLEAN UP CHAR GEN8
37.23                   JPSR            RTN
37.24                   MDAR            SEE
37.25                   JUMP            M0              [G8 GET THE BL8CK
37.26           R4:     JPSR            FN1             [WAIT F8R NEXT J8B
37.27                   JUMP'I          FN7             [RETURN
37.30           FN5:    N90P            [DISPLAY NEXT BL8CK
37.31                   MDAR'L          [SET UP F8R RETURN
37.32                   R5
37.33                   ARMD            FN3
37.34                   MDAR            BUF             [L8AD THE L8CATI8N IN93MEM
37.35                   JUMP            M0              [G8T8 FN3
37.36           R5:     JPSR            FN1             [HANG UP
37.37                   JUMP'I          FN5             [RETURN
37.40           FN6:    N90P            [DISPLAY LAST BL8CK
37.41                   MDAR            R6              [SET JP F8R RETURN
37.42                   ARMD            FN3
37.43                   MDAR'L          [DECREMENT THE L8C CTR
```

```
40.1                    -120
40.2                    MDAE            BUF
40.3                    JUMP            MO              [GO GET THE BLOCK
40.4        R6:         JPSR            FN1             [HANG UP
40.5                    JUMP'I          FN6             [RETURN
40.6        FN4:        NOOP            [CONTINUE EXECUTION
40.7                    MDAR'L
40.10                   00106652        [LOAD A RETURN
40.11                   ARMD            C0D2+6          [STORE IT
40.12                   JPSR            PUT             [OVERLAY 93BUFF
40.13                   JUMP            .               [AND STOP HERE
40.14       TINS:       0               [MORE STORAGE
40.15       NR:         0
40.16                   0
40.17                   35100JH
40.20       STORE:      REPEAT          6
40.21                   0
40.22                   ENDR
40.23       SVIT:       0
40.24       S1:         0
40.25       LNFD:       05032JH
40.26       END:        1JH
40.27       TERMINATE
```

BIBLIOGRAPHY

1. ADAGE Incorporated, Software Operating Instructions, 1969.
2. ADAGE Incorporated, Programmer's Reference Manual, Revision H, 1969.
3. Scientific Data Systems, FORTRAN IV Reference Manual, 1966.
4. Scientific Data Systems, SDS Real Time Monitor, 1967.
5. Scientific Data Systems, SDS SYMBOL and META-SYMBOL, 1967.
6. Scientific Data Systems, SDS 9300 Computer, 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst Professor G. E. Heidorn, Code 55 Hd Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	2
4.	Electrical Engineering Computer Laboratory, Code 52E1 Naval Postgraduate School Monterey, California 93940	1
5.	LT Elton Truxton Ashby, Jr. 601 South Pine Street Little Rock, Arkansas 72205	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

3 REPORT TITLE

The Use of an Auxiliary Computer with a Graphic Display as an On-line Debugging Aid

4 DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Master's Thesis; (June 1971)

5 AUTHOR(S) (First name, middle initial, last name)

Elton Truxton Ashby, Jr.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
	86	6

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10 DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

Often, the most time consuming and costly evolution in the development of computer programs and systems is the testing of the programmer's logic. There are many tools and techniques available which aid the programmer in detecting logic errors, but all have characteristics which limit their usefulness. The objective of this research was to develop a system which uses an ADAGE AGT-10 as an auxiliary computer with a graphic display to provide facilities for monitoring a program which is running on the XDS-9300 computer. The system developed and implemented enables the programmer to stop the execution of his program, display the memory contents of the 9300, change the memory contents as required, and then continue the execution of his program. The thesis includes information on the use of the system and a detailed discussion of the implementation.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
nteractive graphics						
ebugging aids						
n-line error correction